NOVEL COMPUTATIONAL METHODS FOR IMPROVING FUNCTIONAL ANALYSIS FOR LONG NOISY READS

By

Nan Du

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science — Doctor of Philosophy

2019

ABSTRACT

NOVEL COMPUTATIONAL METHODS FOR IMPROVING FUNCTIONAL ANALYSIS FOR LONG NOISY READS

By

Nan Du

Single-molecule, real-time sequencing (SMRT) developed by Pacific BioSciences (PacBio) and Nanopore sequencing developed by Oxford Nanopore Technologies (Nanopore) produce longer reads than second-generation sequencing technologies such as Illumina. The increased read length enables PacBio sequencing to close gaps in genome assembly, reveal structural variations, and characterize the intra-species variations. It also holds the promise to decipher the community structure in complex microbial communities because long reads help metagenomic assembly. However, compared with data produced by popular short read sequencing technologies (such as Illumina), PacBio and Nanopore data have a higher sequencing error rate and lower coverage. Therefore, new algorithms are needed to take full advantage of third-generation sequencing technologies.

For example, during an alignment-based homology search, insertion or deletion errors in genes will cause frameshifts, which may lead to marginal alignment scores and short alignments. In this case, it is hard to distinguish correct alignments from random alignments, and the ambiguity will incur errors in structural and functional annotation. Existing frameshift correction tools are designed for data with a much lower error rate, and they are not optimized for PacBio data. As an increasing number of groups are using SMRT, there is an urgent need for dedicated homology search tools for PacBio and Nanopore data.

Another example is overlap detection. For both PacBio reads and Nanopore reads, there is still a need to improve the sensitivity of detecting small overlaps or overlaps with high error rates. Addressing this need will enable better assembly for metagenomic data produced by the third-

generation sequencing technologies.

In this article, we are going to discuss the possible method for homology search and overlap detection for the third-generation sequencing. For overlap detection, we designed and implemented an overlap detection program named GroupK. GroupK takes a group of short *k*-mer hits, which satisfy statistically derived distance constraints to increase the sensitivity of small overlap detection. For homology search, we designed and implemented a profile homology search tool named Frame-Pro based on the profile hidden Markov model (pHMM) and consensus sequences finding method. However, Frame-pro is still relying on multiple sequence alignment. So we implemented Deep-Frame, a deep learning model that predicts the corresponding protein function for third-generation sequencing reads. In the experiment on simulated reads of protein coding sequences and real reads from the human genome, our model outperforms pHMM-based methods and the deep learning based method. Our model can also reject unrelated DNA reads and achieves higher recall with the precision comparable to the state-of-the-art method.

This thesis is dedicated to my beautiful wife Wenjing and my parents, whose encouragement along the way was paramount to making it thus far.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Dr. Yanni Sun. It was coincidently that I heard there is an opportunity to work with Dr. Sun at 2014 winter, and I almost don't have any bioinformatics background when I first talked to Dr. Sun four and half years ago. But she still patiently discussed with me about possible projects and give me the chance to study the algorithms of sequence analysis under her guidance. She is very nice and patient, never blamed me but instead tried to help me when I made mistakes. Dr. Sun helped me a lot in my research work, courses, and how to write and present research results. Without her help, nothing could be achieved.

Next, I want to thank my Ph.D. guidance committee members, including Dr. Kevin Liu, Dr. James Cole, and Dr. Jiayu Zhou. Dr. Liu gave me many valuable suggestions in my qualifying exam, my comprehensive exam, and my final defense. I am very grateful that he can serve as co-chair of my committee. Dr. Zhou gave me the lecture of Machine Learning and shared his insights on deep learning with me on my final project. Dr. Cole helped me a lot for both my comprehensive exam and final defense.

I would also like to thank National Science Foundation and Summer Fellowship from College of Engineering for funding the research presented herein, and also MSU Dissertation Completion Fellowship (Spring 2019) which allowed me to complete this dissertation. Many thanks to staff at CSE, and especially staff at the High-Performance Computing Center at Michigan State University, for their generally support for all the resources that I need during my Ph.D. study.

I want to thanks my lab mates, including Dr. Jikai Lei, Dr. Prapaporn Techa-angkoon, Dr. Jiao Chen, and Yumeng Wen, for their help on my either research work or daily life. Jikai gave me many useful suggestions when I joined the group. He also helped me a lot in my internship finding. Jiao and I have many discussions on research, daily life, and job finding. I would also

thank all my friends who helped me at MSU in the past seven years. Thanks to my lab mates at UEC lab in Physics, including Dr. Zhensheng Tao, Dr. Kiseok Chang, and Faran Zhou. They always encouraged me for challenges and comforted me when I depressed. Thanks to Chaoyue Liu, who helped me to make the decision that transfers to Computer Science and Engineering. Thanks to Dr. Zhen Zhu and Yanan Chen, who have helped me in both my academic and personal life. Thanks to my friends in the MSU Physics QQ group, Dr. Xu Lu, Dr. Yanhao Tang, Dr. Jie Guan, Dr. Mengze Zhu, and Xukun Xiang. We really have had a good time discussing many interesting topics from frontiers of physics to daily life in the United States. Thanks to Dr. Yuan Gao, Dr. Li Ke, Dr. Xiaoran Zhang, Dr. Shuang Liang, and Dr. Chuanpeng Jiang, for their company from the time when we arrived at East Lansing together. Thanks to my friends and classmates at Computer Science and Engineering who studied, worked, and had fun together with me, including Dr. Jianpeng Xu, Dr. Nan Liu, Dr. Chen Qiu, Dr. Shaohua Yang, Ding Wang, Nan Xia, Wei Wang, Zhuangdi Zhu, Jun Chen, Qi Wang, Kaixiang Lin, Deliang Yang, Tian Xie, Jiawei Li, Sihan Wang, Zhiwei Wang, Yao Ma, and Manni Liu.

Finally, I want to express my sincere gratitude to my wife and my parents. They have always been encouraging and supporting me for pursuing my career, and are always there whenever I need them.

TABLE OF CONTENTS

[1] Du, Nan, Jiao Chen, and Yanni Sun. "Improving the sensitivity of long read overlap detection using grouped short k-mer matches." BMC genomics 20.2 (2019): 190.

[2] Du, Nan, and Yanni Sun. "Improve homology search sensitivity of PacBio data by correcting frameshifts." Bioinformatics 32.17 (2016): i529-i537.

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

Chapter 1

Introduction

Second generation sequencing technologies offered several significant improvements over traditional Sanger sequencing and became an obvious choice for metagenomics analysis. Using shotgun sequencing, people can interrogate the composition and function of microbial communities efficiently. This information can lead to discovery on new biology active compounds, antimicrobials, virulence factors, or metabolic pathways [144]. The traditional downstream method uses reference genomes to annotate the metagenomics dataset. However, in most cases, the metagenomics dataset cannot be corresponded to the known reference genomes. For example, 2 % to 96 % of metagenomics sequence from human skin cannot be annotated by the traditional method, based on the origins of sample [112].

De novo approach, or a reference-free approach, is one alternative method that can be used to overcome the lack of reference. With short reads from the second-generation sequencing platform, *de novo* assembly can generate contigs followed by the taxonomy binning to determine the compositions of the sample. However, with the complex and similar genomes in metagenomics, it is challenging to use short reads from the second-generation sequencing technology to assembly. In such circumstances, long reads from the third generation sequencing platforms have its advantages to generate a better overlap graph for the assembler.

Another alternative method is analyzing the average properties of whole communities rather than focusing on several genomes. Such an approach is irreplaceable as in particular circumstance assembly may not even possible. In this method, longer reads are still preferred as it can provide more accurate information than shorter fragments.

Here, we present the principles of the third generation sequencing technologies, some widely-used algorithms designed for third generation sequencing in genome assembly, the challenge for existing functional analysis algorithms when using third generation sequencing data.

1.1 Third-generation sequencing

Several third generation sequencing technologies were proposed [15, 39]. They share a lot of similar characteristic, like long read length and high error rate. In these techniques, the most intensively studied are the single-molecule, real-time sequencing (SMRT), developed by Pacific BioSciences (PacBio), and Nanopore sequencing, developed by Oxford Nanopore Technologies (Nanopore). Here we will introduce the theorem of SMRT and Nanopore sequencing technologies. Although some algorithms we discussed here were originally proposed to SMRT, it can also be used for Nanopore sequencing [93].

1.1.1 SMRT

To achieve real-time sequencing at the single molecule level, SMRT adopted an approach to generate sequencing information based on the sequential fluorescent signal generated during DNA synthesis by a polymerase molecule.

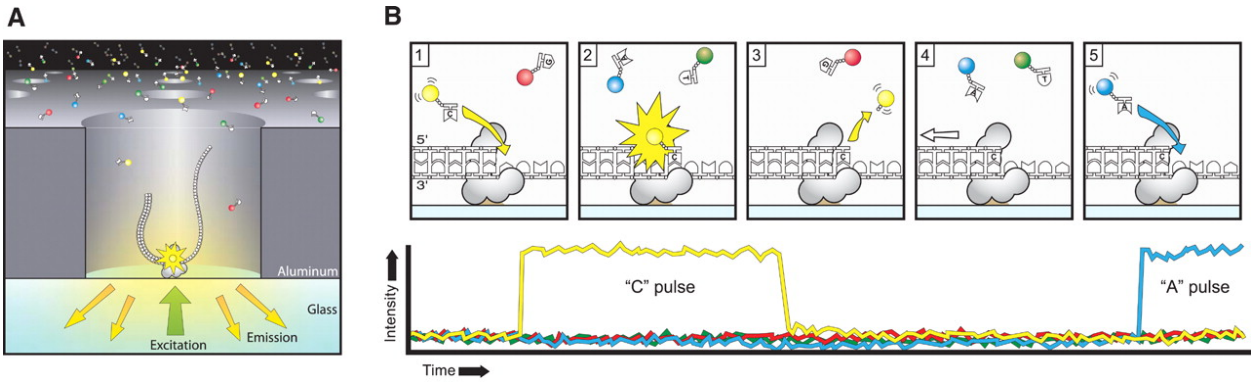

Figure 1.1: Principle of single-molecule, real-time DNA sequencing. Reprinted from [39].

The basic sequencing unit in SMRT is a zero-mode waveguide (ZMW). The propagation mode in the waveguide will limit the wave, satisfying the mode to propagate in the waveguide (precisely determine the frequency and form). The "zero-mode waveguide" means the waveguide provides zero guided modes for the propagation of the electromagnetic wave. Observation in the zero-mode waveguide can reach single fluorophore event scale even in the high concentration of fluorescent molecules present [81]. In the bottom of each ZMW there is located a single molecule of DNA template-bound Φ 29 DNA polymerase.

During an observation, a phospholinked nucleotide forms a hydrogen bond with the nucleotide in the template. This process will produce a fluorescence signal at the bottom of ZMW. The light pulse will end upon the cleavage of the dye-linker-pyrophosphate group. By detecting the light pulse and recording a series of movies of the signal, we can have all information which later can be translated to bases in continuous long reads (CLR). The average CLR read length from Pacbio RSII using C4 chemistry can reach more than 10k bps [145], with an 11 % to 15 % error rate [8,80].

Using the circular template, SMRT can sequence DNA multiple times. By doing this, a high quality read called circular consensus sequence (CCS) can be generated with the more than 99 %

accuracy with 15 X coverage. However, the trade-off of CCS is the short-read length (averagely is about 1k bps [114]) due to the limitation lifetime of the DNA polymerase [134].

1.1.2 Nanopore sequencing

Oxford Nanopore is another promising third-generation sequencing technology. In Nanopore sequencing, a DNA strand is threaded through a nanopore, and the ionic current fluctuations can be used to identify the corresponding DNA sequences.

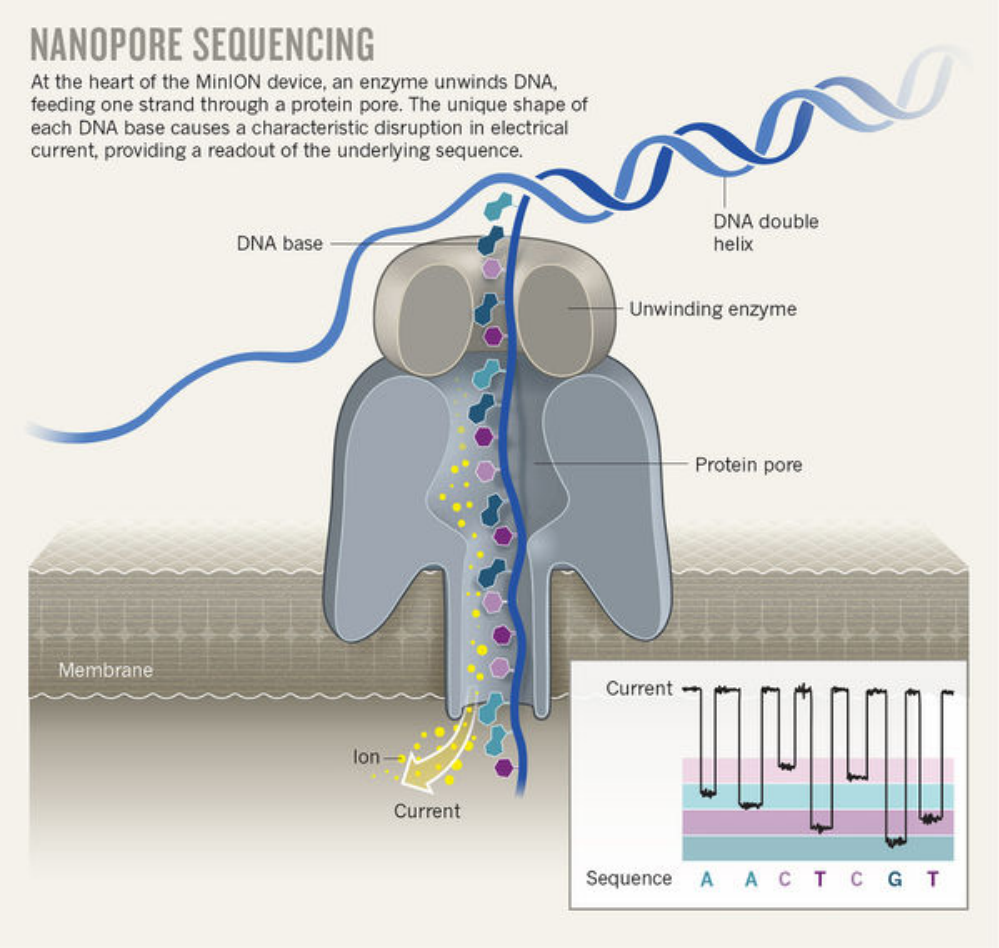

Figure 1.2: Principle of nanopore sequencing. It determine the type of nucleotide by measuring the current level of the base passes passing through the nano hole. Reprinted from Naturejobs website.

In theory, the nano holes can be created by proteins puncturing membranes (biological nanopores)

or in solid materials (solid-state nanopores). In Nanopore sequencing, the nanopore is created utilizing the heptameric protein α-hemolysin (α HL) [28]. During the sequencing, DNA will bind to the binding site of the nanopore. The sequencer reads the disruption in the current passed through the membrane as each base passes through. Since the bases have different electronic characteristics, each nucleotide has a different signature of disruption, allowing for base calls.

Nanopore sequencing can produce very long sequences that length up to 2 million bases in theory. In a recent experiment with MinION R9.4 1D chemistry, people were able to sequence ultra long reads with 882,000 bp, with a median length of 10,589 bp. The error rates of Nanopore sequencing are close to PacBio, which are around 18% [67].

1.2 Sequence analysis algorithms

In this section, we overviewed the common sequence analysis algorithm and its application in third-generation sequencing reads.

1.2.1 Alignment

Alignment program is one of the fundamental algorithms for Pacbio SMRT application, as right now the error correction and whole genome assembly both rely on building the alignment first.

The fundamental idea of aligning is using dynamic programming and selecting a maximal score path given the different penalty of insertion, deletion, and mismatch (Figure 1.3). For example, in BLAST [4], an identity will score 5 and a mismatch will get penalty of 4. Several different approaches are adopted to accelerate the performance of aligning. For example, the "seed and extend" method finds exact matches in k-mer scale first and then reduces the search space by only

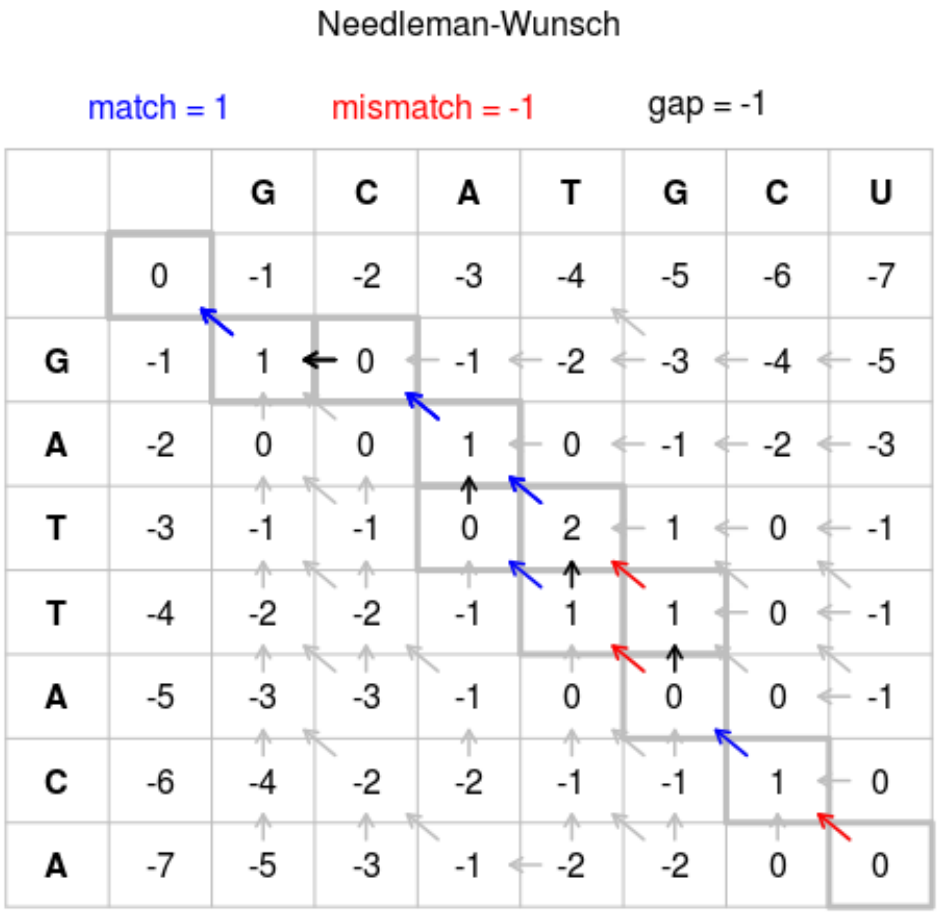

Figure 1.3: A dynamic programming alignment matrix for aligning sequence `GATTACA` against `GCATGCU` using Needleman-Wunsch algorithm [105]. In this example, a simple scoring system, match = 1, mismatch = -1, gap = -1, is used. (a) Start from the cell in the second row, second column. Move through the cell row by row, calculating the score for each cell. (b) The score is the maximal score calculated from the cell from the top, left, or left-top. If the score is calculated using left-top cell, means a match or mismatch. Otherwise is an insertion or deletion. (c) After finishing the filling of all table, the score of the right-bottom cell of the matrix is the score for global alignment of the two sequence. (d) Traceback from the last cell is needed to obtain the alignment. The final alignment is `G-ATTACA` vs. `GCA-TGCU`. Reprinted from Wikipedia.

allowing the sequence to keep with the number of matches of k-mers above the threshold. Other methods like Burrows-Wheeler Transform [83, 84] and Hashing function [90] can be also applied to traditional alignment tools. Although people can use those tools on Pacbio SMRT reads, the sensitivity, accuracy and speed of alignment are compromised due to the long erroneous reads.

Right now, BLASR [21] is a popular alignment tool designed for long reads with high error rates. BLASR combined data structure used for short read mapping and the alignment method for whole genome mapping. At first, the program will cluster the short exact matches found using the BWT-FM index [42, 84] or suffix array [97]. By doing this, the program can determine the approximate coordinate that reads should align to the genome. A rough alignment using the sparse dynamic programming on the set of short exact matches will be the following steps. Finally, with the guide of the sparse dynamic programming, an accurate alignment will be generated using dynamic programming. Compare with other tools like BWA-SW (aligning 132M bases in 434 minutes) [89] and BLAT (aligning 181M bases in 4724 minutes) [73], BLASR can align the largest number of reads (230M bases) to the reference genome with the fastest speed (20 minutes) using an *E. coli* O104: H4 dataset.

There are more tools designed for third generation sequencing coming. For example, GraphMap [137] is another tool designed for third generation sequencing with a gapped-seed strategy. We will discuss alignment tools for third generation sequencing later in detail.

1.2.2 Error correction

Error correction is a necessary step for using SMRT sequencing as the error rate of raw reads is too high. However, longer reads with small bias make consensus error correction possible. Right now, two major approaches are available: hybrid [8, 55, 58, 78, 99, 130] and non-hybrid [24, 93] error correction.

Hybrid error correction uses short but accurate second generation reads to help correct error in

long third generation reads. The first type of hybrid error correction tools (LSC [8], PacBioToCA [78], and porovread [55]) aligned accurate short reads to third generation long reads, and then using the consensus algorithm to generate error-corrected consensus sequences. The second type of tools (LoRDEC [130], Jabba [99]) exploits the context in short reads by build a de Bruijn graph first from those short reads. Long reads from Pacbio SMRT are then added as the guide to help the program decide the best path in the de Bruijn graph. Other methods [58] beyond these two types are also available, but still share a lot of similarity to the tools mentioned above.

Unlike hybrid error correction which needs second generation reads like Illumina, non-hybrid error correction methods only use reads from the third generation sequencing. HGAP(hierarchical genome assembly process) [24] used non-hybrid error correction before starting the assembly process. The method first selects the longest seed reads from Pacbio SMRT raw reads. Then aligned other "short" read to those seed reads and built a multiple sequence alignment graph. The program can easily find the maximal score path in the graph using dynamic programming. After the pre-assembly stage, the average accuracy of the dataset is increased from 86.9 % to 99.9 % [24]. Similar methods [93] were also applied to reads from Oxford Nanopore successfully. The nonhybrid method relies on the coverage of dataset [34].

1.2.3 Whole genome assembly

The long reads from third generation sequencing is more favorable for whole genome assembly as long reads can overcome many limitations from short reads, such as handling highly repetitive genomic regions.

The non-hybrid *De novo* assembly [24] needs pre-assembly error correction step to compensate

the high error rates of Pacbio data. High quality reads after error correction are feasible to apply traditional assembly method that can take long-read input. In this case, overlap-layout-consensus (OLC) method is better than the De Bruijn Graph (DBG), although the latter one is quite popular for the second generation sequencing data. In the OLC method [101, 103], the assembly will first try to find all versus all match and then determine if the read pair has overlap. Then a graph will be constructed by treating each read as a node and connecting overlap reads by edges. An assembly algorithm is available by considering all the information in the graph and produces consensus output. Given enough coverage, the quality of HGAP assembly can reach beyond 99.999% (QV 50).

Fast overlapping method is desired for OLC method assembler to reduce the huge computational cost for calculating the overlap by find all-versus-all alignment. The time-consuming overlapping is the bottleneck of *De novo* assembly using SMRT and prevents the application for larger genome [12]. MinHash Alignment Process (MHAP) is a newly developed algorithm can efficiently detect overlap between erroneous long reads. MinHash [16] is the core method of the algorithm that used to estimate the similarity of the two sequences without a complete alignment (Figure 1.4). To evaluate the similarity of the two sequences, the MinHash start by convert all kmers in each of the sequences to integer fingerprint using multiple randomized hashing functions. Then the algorithm will collect the minimal value kmer for each hashing function to construct the sketch of the sequence. The size of the sketch is determined by the number of hashing function and is much smaller than the size of all kmers. The Jaccard Similarity of the sketch of the two sequences can be treat as an approximation of the shared number of kmers between two sequences. This is a computational efficiently to estimate the similarity of two sequences without alignment. In the test, MHAP method (3.6 CPU hours for E. coli K12 dataset)is much faster than the BLASR (87 CPU hours)to find overlapping for assembly. In the experiment, MHAP achieved a comparable or

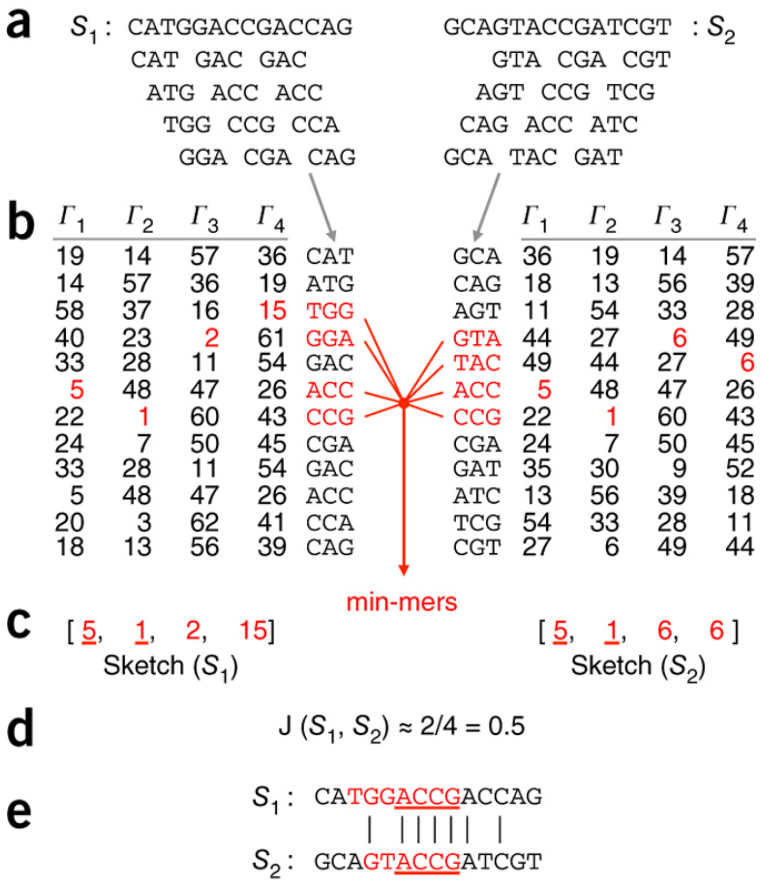

Figure 1.4: Rapid overlapping of noisy reads using MinHash sketches. (a) To create a MinHash sketch of a DNA sequence S, we first obtained all the k-mers of the sequence k-mers. In the example above, $k = 3$, generate 12 k-mers each for S1 and S2. (b) Multiple hash functions are used to convert k-mers to integer fingerprints. The number of hash functions H determines the resulting sketch size. Here, we choose $H = 4$. The minimum kmer which hashing function value is minimal for each hash is referred as min-mer. (c) The sketch of a sequence is composed of the H min-mer fingerprints in order, which is much smaller (size 4) than the set of all k-mers (size 12). In this example, the sketches of S1 and S2 share two same minimum fingerprints (underlined). (d) Real Jaccard similarity (0.22) is estimated using the fraction of shared min-mer (0.5) between the sketches. To have an accurate estimation, $H \gg 4$ is needed. (e) If the similarity meets the threshold, the position of shared min-mers in the original sequence is computed to determine the overlap offset of the S1 and S2. Reprinted from [12].

improved assembly than BLASR with less time.

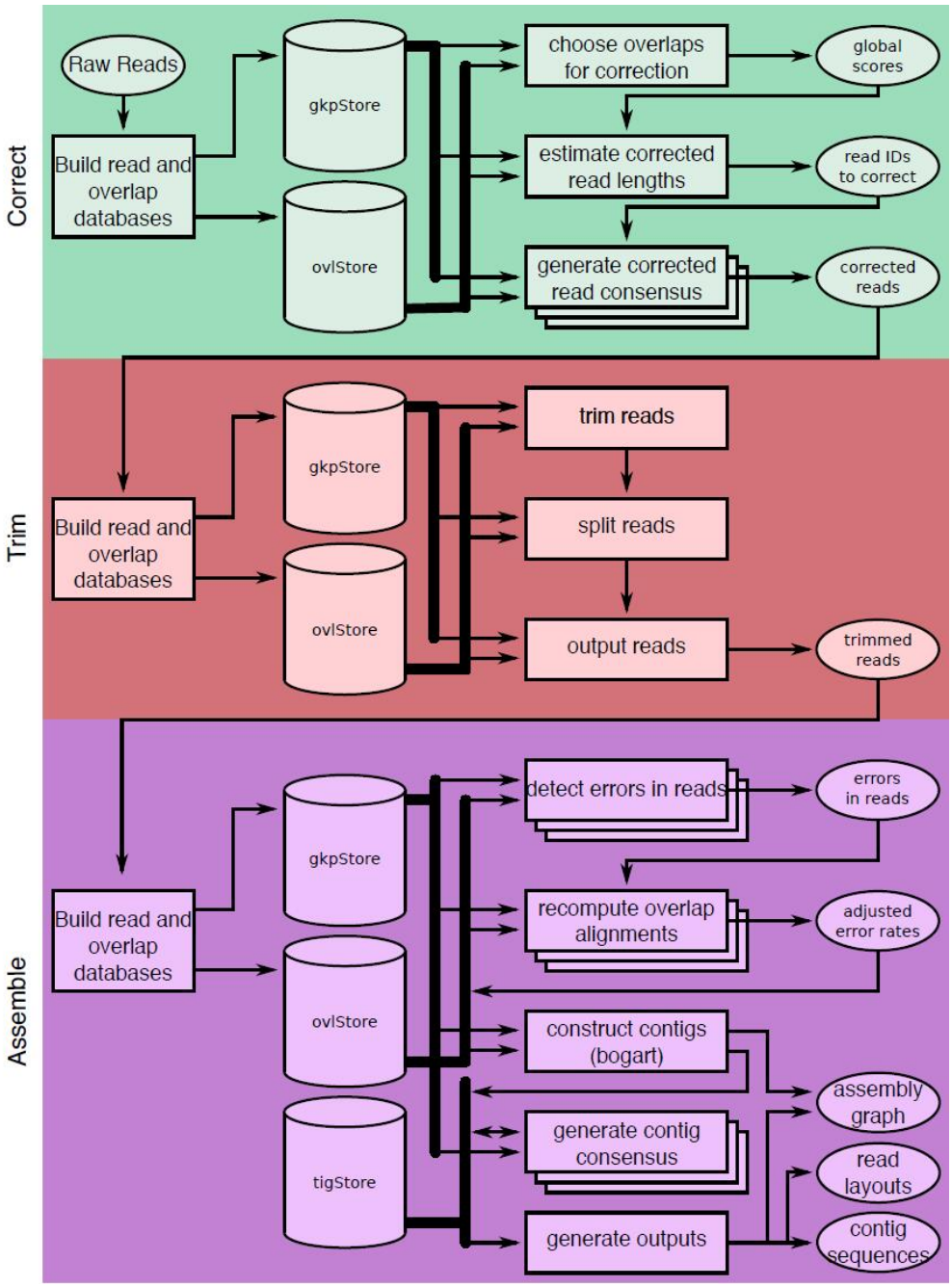

Figure 1.5: Three stage (correction, trimming, and assembly)in a Pacbio SMRT assembler (Canu). The correction step selects the best overlaps to use for sequence correction and generates corrected reads from consensus sequences. The trimming step identifies unsupported regions in the input and trims or splits reads based on the longest supported range. The assembly step makes a final pass to determine sequencing errors; recompute overlap alignments; and outputs contigs, an assembly graph, and summary statistics. Reprinted from [79]

1.3 Challenges and overview

Although many novel algorithms for alignment, error correction, and whole genome assembly have been developed for handling long noisy reads, there are still many challenges for the third-generation sequence analysis. First, most novel algorithms are designed to target complete genome assembly. However, in many scenarios, like metagenomics and transcriptomic sequencing, the whole genome assembly is not realistic, or the result of the assembly is not satisfied given the coverage of sequencing. In both cases, downstream analysis algorithm needs to process long read with high error rates. For example, for homology search and protein domain annotation, current methods like pHMM cannot handle the frameshifts introduced by a large number of insertions and deletions in the third-generation sequencing reads, leading to short or non-significant alignments in the downstream analysis. Second, for applications like metagenomics, there may coexist many similar protein domains in the dataset, which is not easy to distinguish, even without errors. Third, existing algorithms designed for PacBio or Nanopore still have the potential for improvement in performance and efficiency.

In order to address these challenges and improve the performance of sequence analysis, especially protein domain prediction and annotation, we have proposed and developed three tools: GroupK, FramePro, and DeepFrame. GroupK is an overlap detection tool designed for PacBio and Nanopore data. The experimental results showed that GroupK enables more sensitive overlap detection, especially for datasets of low sequencing coverage. Both FramePro and DeepFrame are designed for improving protein prediction from third-generation sequencing. FramePro is a profile homology search tool designed for PacBio reads. It utilizes both profiles hidden Markov model and sequence alignment graph consensus to enable more sensitive homology search. DeepFrame focuses on identifying encoded protein domains from a single long noisy DNA read. It uses

a convolutional neural network to extract useful features to distinguish protein-coding sequences automatically. In the experiment, it outperforms pHMM-based methods in both classifications on multiple protein families, and the detection of protein domain from other unrelated DNA reads.

Chapter 2

Overlap Detection On Long Noisy Reads [1]

Genome assembly using third-generation sequencing data requires dedicated methods and tools. Existing genome assembly tools mainly utilize two types of graph models: overlap graph and de Bruijn graph. When the error rate is low, de Bruijn graph has the theoretical advantage that the graph size does not increase significantly with the sequencing coverage, which is usually high for Illumina datasets. For third-generation sequencing data, the high error rate and low coverage make the overlap graph a sensible choice for genome assembly [78]. A key step in constructing the overlap graph is to identify read pairs that share overlaps, which indicates that these reads are sequenced from the same loci in the underlying genome. Although there are a number of sequence alignment programs available for conducting overlap alignment [4, 132], a majority of them rely on dynamic programming and are too computationally expensive for high throughput sequencing data. Due to high error rates, existing short read overlap detection software using BWT (Burrows-Wheeler transform) or hash table [51, 135] cannot be directly applied to long reads. To address this challenge, many new method was proposed. Here, we summarize the major existing method and also introduce our new method for detecting overlap from long noisy sequences.

[1] Du, Nan, Jiao Chen, and Yanni Sun. "Improving the sensitivity of long read overlap detection using grouped short k-mer matches." BMC genomics 20.2 (2019): 190.

2.1 Background

2.1.1 Related work

Two strategies are currently being employed to detect overlaps for error-prone long reads. One strategy tries to correct sequencing errors in PB (Pacific Biosciences) and ONT (Oxford Nanopore) data before overlap detection. There exist a number of sequencing error correction tools [24, 78]. Some of them rely on hybrid sequencing, which requires preparation of at least two sequencing libraries and several types of sequencing runs and thus is not cost-effective for many applications. Others conduct error correction using long reads only. One representative method is described in Chin et al.'s hierarchical genome-assembly process HGAP [24], whose performance improves with higher read coverage. It is worth noting that for alignment-based error correction methods such as the one in HGAP, an important step is to identify reads that can be aligned quickly. Essentially, techniques used for overlap detection can be used for alignment detection as well.

The second category bypasses the difficulty of error correction and identifies overlaps using raw reads. Various approximate similarity search methods have been applied on PB and ONT data [26]. They generally follow seed-chain-align procedure [88]. Seed-based filtration step plays an essential role in controlling the trade-off between sensitivity and computational efficiency. Usually, these methods use short string matches as the filtration step. A short string or k-mer match requires exact matches of k consecutive characters between two sequences. Intuitively, overlapping reads tend to share more common k-mers than non-overlapping reads. Strategies that can quickly find the number of shared k-mers can thus be applied. In this section, we summarize the main strategies of several state-of-the-art overlap detection tools. We highlight the differences between our method and the existing ones in the following section.

MHAP [12], Minimap [87, 88], and DALIGNER [102] all use k-mer matches for identifying

candidate overlapping pairs. Due to the high error rate, usually only short k-mers will be applied in order to achieve high sensitivity. However, identifying short k-mer matches between all pairs of reads is computationally expensive. Thus, the leading tools employed different data structures and algorithms for estimating k-mer-based similarity. MHAP converts long reads into sets of k-mers and sketches using minHash. Then the similarity between reads is estimated using the compact sketches. Minimap also uses a compact representation of the original reads by keeping minimizers rather than all possible k-mers of a read. Then collinear k-mers will be clustered and used for checking possible overlaps. DALIGNER directly sorts k-mers based on their positions and then utilizes merge sort to identify the number of shared k-mers. As the sorting is cache efficient, DALIGNER is practically very efficient.

BLASR [21] was initially designed for mapping PB reads to a reference genome. It is also widely used as an overlap detection tool for PB data. BLASR uses BWT and FM index to identify short k-mer matches and then clusters k-mer matches within a given distance range. The clustered k-mers are ranked based on a function of the k-mer frequency. Only highly ranked clusters will be kept for downstream analysis.

Being different from the above tools, GraphMap [137] uses spaced seeds that allow matches of non-consecutive characters. Spaced seeds were initially used in homology search for improving the trade-off between sensitivity and filtration efficiency [18, 95, 139]. In particular, spaced seeds containing the pattern "11*" have high sensitivity in capturing homologous protein-coding genes because of the codon structure. However, designing optimal spaced seeds (i.e., deciding the positions of the wildcard characters) is NP-hard [94, 106]. GraphMap empirically chooses two spaced seeds. Ideally, different sets of seeds may be designed for input data of different error profiles.

2.1.2 Overview of our work

The high error rates and also the different error profiles of PB and ONT data motivate us to use a more flexible seeding strategy called group hit criteria [109], which define a group of possibly overlapping *k*-mers satisfying statistically derived distance constrains. For brevity, we will call the *k*-mers set satisfying the group hit criteria as a "group seed". A group seed was initially proposed and used for homology search. Given the error profiles, such as the estimated indels and mismatch probabilities, thresholds for grouping short *k*-mers can be computed using the waiting time distribution and the one-dimensional random walk [109]. A group seed can effectively handle all types of errors and is ideal to detect small overlaps. With group seeds, we can achieve high sensitivity using short *k*-mers (e.g., 9-mer) while still maintaining a desirable specificity.

In this work, we employ group seeds for detecting overlapping long reads for *de novo*genome assembly. Our implementation, named GroupK, provides a complementary tool to existing methods for detecting small overlaps or overlaps compounded by high error rates of both reads. This ability enables our tool a sensible choice for genome assembly in metagenomic data sequenced by third-generation sequencing platforms. As these community samples usually contain microorganisms with heterogeneous coverage, being able to identify small overlaps will be very important for reconstructing genomes of rare species.

2.2 Methods

GroupK is designed for improving the sensitivity of detecting small overlaps or overlaps with low identity. Currently, third-generation sequencing data still has high error rates. The overlapping regions formed by two error-prone long reads can have lower sequence identity than mapping a long read against a reference genome. Figure 2.1 presents the histogram of the overlap size and the

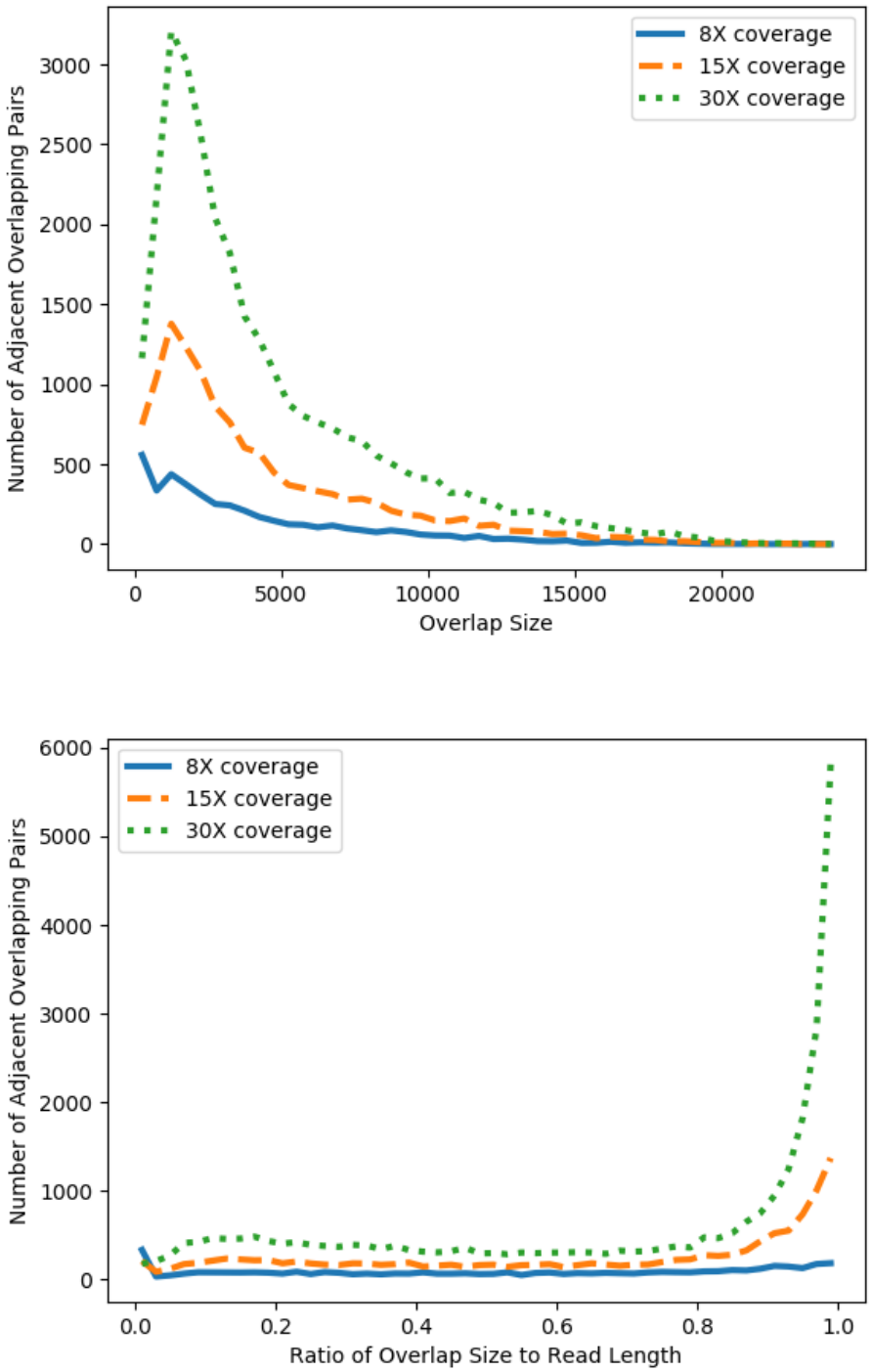

Figure 2.1: Histograms of irreducible overlap sizes (**top**) and the ratio of overlap size to the read length (**bottom**) when comparing adjacent overlapping reads on simulated PB *E. coli* datasets given different coverages. The bin width for the overlap size is 500. For the 30X dataset, the average read length is 8366 and the number of reads is 16644. For the 15X dataset, the average read length is 8253 and the number of reads is 8436. For the 8X dataset, the average read length is 8414 and the number of reads is 4413.

corresponding ratio of overlap size to the read length between two adjacent reads. The reads are simulated using PBSIM [114] from *E. coli* with three different coverages. As we know the position of each simulated read in the genome, the overlap size can be easily decided. Note that a read can form overlaps with multiple reads sequenced from the same region. However, the two figures are generated using overlaps between two adjacent reads, which define an "irreducible" edge [100] in an overlap graph. Thus, these overlaps can decide the continuity of the final genome assembly. The figures show that there are still substantial regions with small overlaps. For example, there are 45.46%, 34.76%, and 31.19% of the overlaps shorter than the 50% of the read length for data with coverage of 8X, 15X, and 30X, respectively. It will be ideal to detect relatively small overlaps to fully take advantage of the long reads for generating more complete assemblies.

2.2.1 Pipeline

Identifying small overlaps is computationally difficult. Thus, we use a carefully designed hierarchical filtration strategy to distinguish true overlapping reads from non-overlapping ones. The pipeline of GroupK consists of three key steps: filtration, group seed matching, and chaining (Figure 2.2). Filtration is used to reduce the search space by quickly identifying read pairs sharing a minimum number of k-mers. High insertion/deletion error rates tend to produce short k-mer matches on different diagonals. Thus, we adopt group seed matching to identify a group of short k-mer matches in close proximity. There are two types of distance constraints. 1) The distance (number of nucleotides) between the k-mers on x-axis and y-axis must be smaller than a given threshold; 2) the diagonal distance, which is the difference of the diagonals of two k-mer matches, must be within a given range. Chaining is used to estimate the final overlap region. Figure 2.3 shows that applying group seed can remove a large number of random k-mer hits while keeping the k-mer matches within the overlapping region. When $k = 15$, there are only two hits, and it is

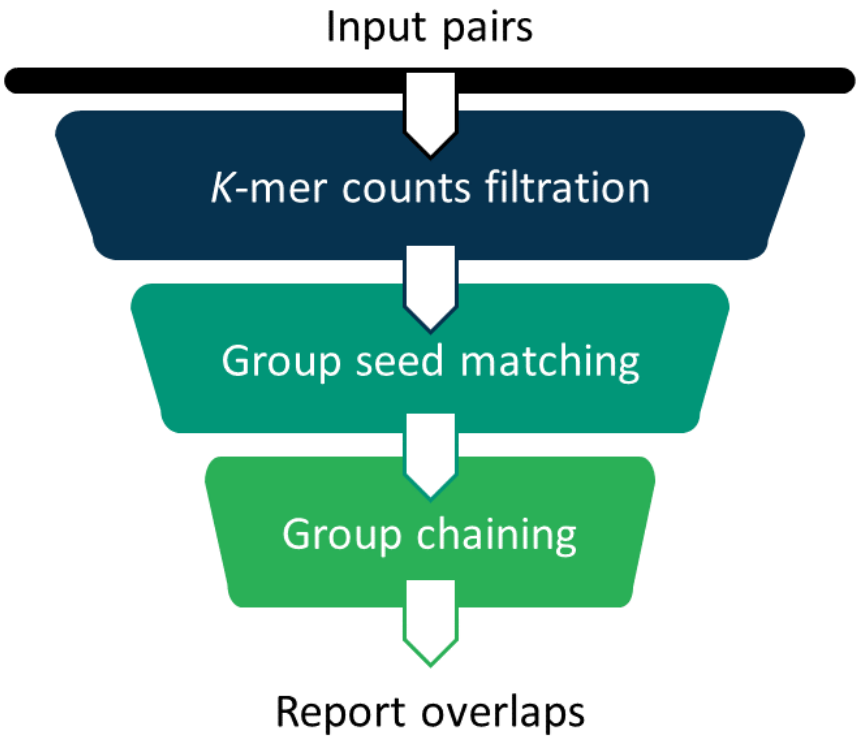

Figure 2.2: The pipeline of GroupK.

difficult to determine whether there is an overlap. When $k = 9$, there is clearly a chain formed by hits in the overlapping region. However, there are also a large number of random hits. With group seed matching criteria, most of the random 9-mer hits are filtered out. So the downstream analysis becomes more straightforward.

2.2.2 Estimate the expected number of k-mers for the filtration stage

In this section, we analyze the expected number of random k-mer hits between two reads and also k-mer hits in overlaps. The analysis will be used for determining the k-mer size and also other parameters for the filtration stage. Given two reads (S_1 and S_2) with length L and error rate ε, we want to determine how many k-mer hits we expect to find between S_1 and S_2.

We first consider the case that S_1 and S_2 are not related (no overlap). By assuming that the bases in S_1 and S_2 are randomly distributed, the expected number of random k-mer matches $E[X_r]$ is roughly:

$$E[X_r] = \left(\frac{1}{|\Sigma|}\right)^k \cdot L^2 \tag{2.1}$$

Note that this equation is different from the expected number of shared k-mers in MHAP [12] because we distinguish k-mer hits based on their locations rather than the k-mers themselves. Also, we assume that overlapping k-mers are independent.

In the second case of S_1 and S_2 forming an overlap, we estimate the expected number of k-mer matches in the overlap by first computing P_o, which is the probability of observing a k-mer match within the overlap. In the case of no sequencing error (i.e. $\varepsilon = 0$), the probability of observing a k-mer match at an aligned position in an overlap is simply 1.0. But in the practical case of $\varepsilon > 0$, we need to consider two scenarios in order to determine the probability of observing two identical characters at an aligned position in the overlap: 1) the characters from S_1 and S_2 are correct; 2) the

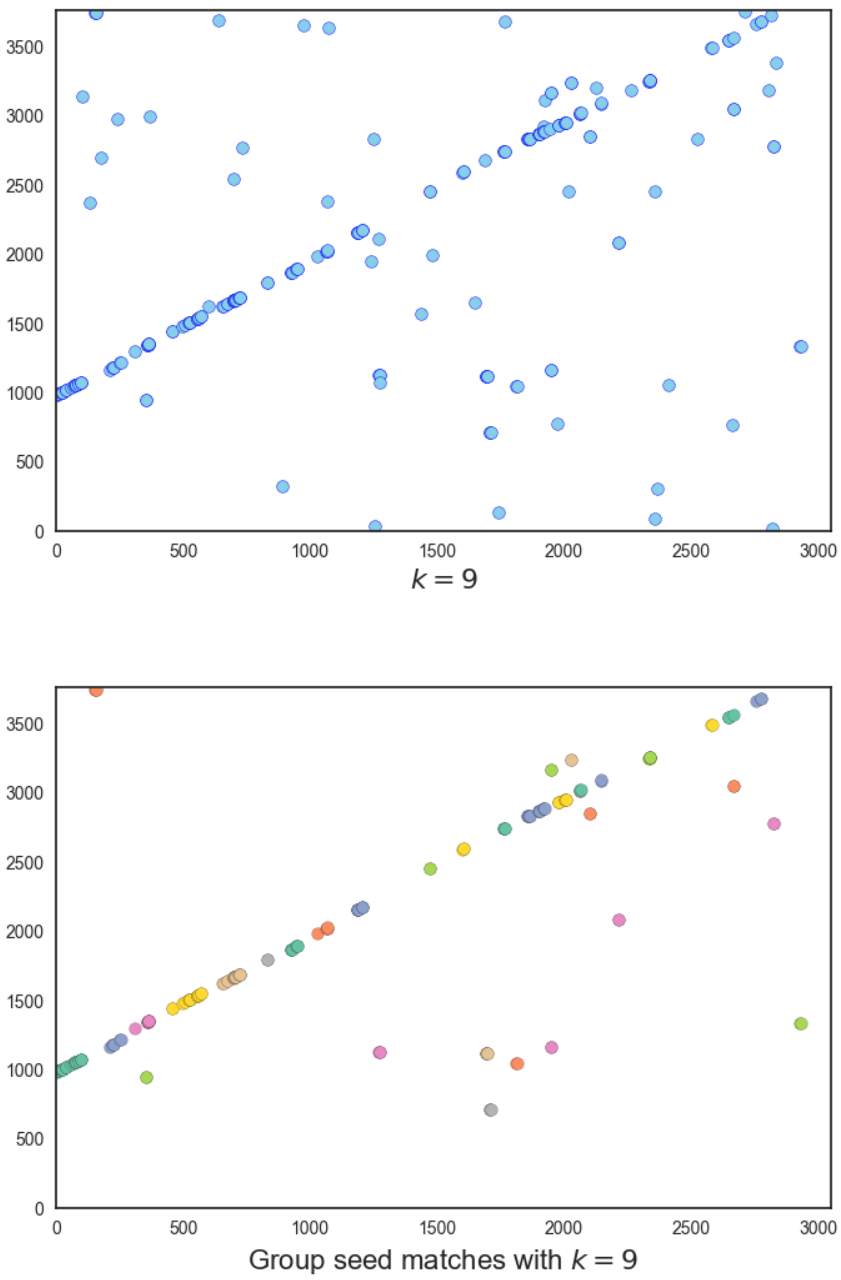

Figure 2.3: The dot plot of k-mer hits and group seeds matches for one overlapping pair from the *E. coli* PB dataset used in Section 2.3. Each dot is a k-mer hit. The x-axis and y-axis show the locations of the hits on the reads. The overlap region is roughly from 0 to 2800 on the x-axis, and from 1000 to 3800 on the y-axis. **Top:** all 9-mer hits. **Bottom:** 9-mer hits that passed the group hit criteria. A group seed is represented by closely located dots of the same color.

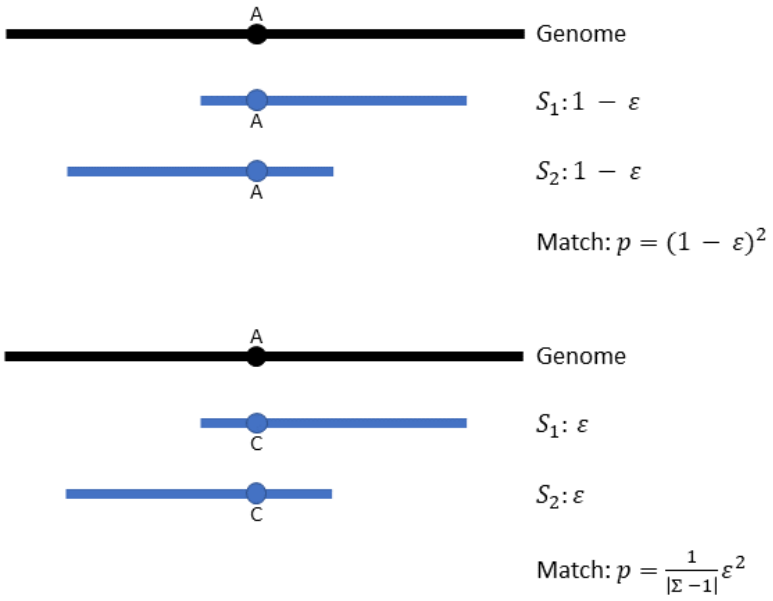

Figure 2.4: Two cases contributed to the identical characters at an aligned position in the overlapping region of S_1 and S_2. S_1 and S_2 are two reads sequenced from the same region of the underlying genome and form an overlap. **Top:** both bases on S_1 and S_2 are correct, forming a match. **Bottom:** both bases on S_1 and S_2 are sequencing errors, and substituted by the same character.

two characters from S_1 and S_2 are errors and are randomly substituted by the same character. The two cases are visualized in Figure 2.4. So the probability is given by [12]:

$$P_o = \left[(1-\varepsilon)^2 + \varepsilon^2 \frac{1}{|\Sigma| - 1}\right]^k \tag{2.2}$$

Considering both random k-mer matches and k-mer matches in an overlap, the expected number of shared k-mers between two overlapping reads is estimated by:

$$E[X_o] = P_o \cdot M + \left(\frac{1}{|\Sigma|}\right)^k \cdot L^2 \tag{2.3}$$

M is the size of the overlap. Note that the above equation slightly over-counts the number of k-mer hits in an overlap because the random k-mer hits inside the overlap may be counted twice with

probability $P_o \cdot (\frac{1}{|\Sigma|})^k$. (Also, we assume that the probabilities of substitution and insertion/deletion are on the same order and thus do not distinguish them in the above equation.)

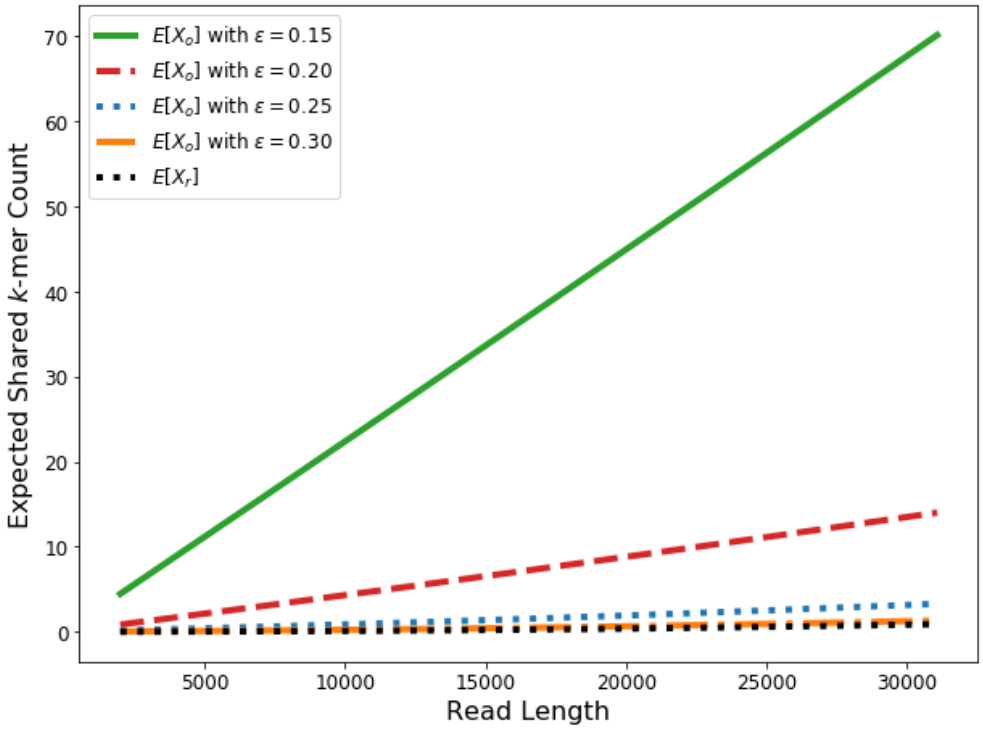

Figure 2.5: The change of $E[X_o]$ and $E[X_r]$ (y-axis) with the increase of the read length (x-axis), which is obtained from a real PB dataset. The overlap size is set as the 1/4 of the read length as we focus on identifying the hard case of small overlaps. $k = 15$.

As we are mainly interested in finding small overlaps or overlaps with low sequence identity, we plot the expected number of k-mer hits with the overlap size being 1/4 of the read size in Figure 2.5. In order to plot the figure, we compute $E[X_o]$ and $E[X_r]$ using the read lengths from a 15X *E. coli* PB dataset (the data of our second experiment in Section 2.3). We only consider reads of length above 2,000. These figures allow us to choose the appropriate threshold for k-mer-counting based filtration. For example, Figure 2.5 shows the expected number of 15-mers between reads of different error rates. $E[X_r]$ started as 4 when $\varepsilon = 0.15$. And, for larger ε, $E[X_r]$ is even smaller. Thus, our default filtration threshold is two 15-mers in order to ensure high filtration sensitivity. The implementation details of the k-mer counting stage can be found towards the end

of the Section 2.2.

2.2.3 Group Hit Criteria

Sequencing errors tend to produce short k-mer matches. In addition, the insertion/deletion errors lead to k-mer matches on different diagonals. Thus, instead of using relatively long k-mers (such as 15 or 16-mers) as existing tools do, we use a group of short k-mers (such as 9-mer) to accommodate the high insertion or deletion error rates. A group seed is a set of possibly overlapping k-mer hits with statistically calculated constraints. The region containing group seeds is more likely to be inside an overlap than a single k-mer hit. Reference [109] first introduced the group hit criteria and also derived the method to calculate the criteria statistically. We apply their method for overlap detection.

Assume that we have two reads S_1 and S_2 of length m and n, respectively. The numbers of k-mers at different positions in S_1 and S_2 are $m-k+1$ and $n-k+1$, respectively. A k-mer hit at position (i, j) is defined by $S_1[i \dots i+k-1] = S_2[j \dots j+k-1]$, where $i \leq m-k+1$ and $j \leq n-k+1$. For two k-mer hits at (i_1, j_1) and (i_2, j_2), their inter-seed distance $D((i_1, j_1), (i_2, j_2))$ is the maximum of $|i_2 - i_1|$ and $|j_2 - j_1|$. The k-mer diagonal of a k-mer hit at (i, j), $d(i, j)$, is defined as $j - i$.

With these notations, the goal is to solve the following inequalities given confidence level $1-\alpha$ defined by significance level α:

$$D((i_1, j_1), (i_2, j_2)) \leq \rho \tag{2.4}$$

$$|d(i_1, j_1) - d(i_2, j_2)| \leq \delta \tag{2.5}$$

ρ and δ are integers we need to define the group hit criteria. For example, when $\alpha = 0.05$, our

goal is to derive ρ and δ so that with 95% chance the inter-seed distance and the diagonal shift between two k-mers in overlapping reads are at most ρ and δ, respectively.

2.2.3.1 Constraint on k-mer distance

We followed the model described in the article [109]. Runs of head in n-independent Bernoulli trials are used to model k-mer matches, with the probability p for a match and $(1-p)$ for a mismatch. In this model, a k-mer match can be treated as k consecutive match runs with probability of p^k. From the waiting time distribution [2, 11], the probabilities of the inter-seed distance x of two k-mers in an overlap are:

$$\mathscr{P}[D_k = x] = \begin{cases} 0 & \text{for } 0 \leq x < k \\ p^k & \text{for } x = k \\ (1-p)p^k(1 - \sum_{i=0}^{x-k-1} \mathscr{P}[D_k = i]) & \text{for } x > k \end{cases} \tag{2.6}$$

With the confidence level $1-\alpha$, ρ can be solved using following equation:

$$\mathscr{P}[D_k \leq \rho] = 1 - \alpha \tag{2.7}$$

In the actual implementation we use $\alpha = 0.05$. Thus, there is 95% chance that the inter-seed distance of the two seeds in an overlap is less than the ρ calculated using Equation (2.7).

2.2.3.2 Constraint on k-mer diagonal distance

The diagonal shift between two k-mers, $|d(i_1, j_1) - d(i_2, j_2)|$, in two overlapping reads is caused by insertions and deletions. Note that the insertions and deletions are defined by comparing two reads, not between a read to a reference genome. So the insertion rate and deletion rate are treated equally.

The diagonal shift between two k-mer hits can be modeled by a discrete one-dimensional random walk model [41, 109]. The diagonal shift starts from 0. Let the steps of the random walk be l. Assume that the insertion and deletion rate is q across the whole read. Thus, the probability of a diagonal change is q, and the probability of staying in place is $1-2q$. Also, we assume that in l steps, there are n_i inserted nucleotides (increase shift), n_d deleted nucleotides (decrease shift), and n_m matched nucleotides (no impact on shift). If the final diagonal shift is i, we have the following equations:

$$\begin{cases} n_i + n_d + n_m = l \\ n_i - n_d = i \end{cases} \tag{2.8}$$

Reference [109] calculated the probability of obtaining a diagonal shift i after l steps in the random walk. According to Equation (8), we have $n_i = i + n_d$ and $n_m = l - (i + 2n_d)$. For a specific n_d, the probability of a random walk producing diagonal shift i can be calculated as the number of the possible paths $\binom{l}{i+2n_d} \cdot \binom{i+2n_d}{i+n_d}$, times the probability product of all insertions, deletions, and no shift change at each step $q^{n_d} q^{n_d+i} (1-2q)^{l-(i+2n_d)}$. To calculate the probability of generating a diagonal shift i given l, $P[i,l]$, we need to consider all possible values of n_d, which is from 0 (no deletion) to $(l-i)/2$ (no match). So we have:

$$\mathscr{P}[i,l] = \sum_{n_d=0}^{(l-i)/2} \binom{l}{i+2n_d} \cdot \binom{i+2n_d}{i+n_d} \cdot q^{n_d} q^{n_d+i} (1-2q)^{l-(i+2n_d)} \tag{2.9}$$

To calculate δ, we sum up the probabilities $\mathscr{P}[i,l]$ for $i = 0, \pm 1, \pm 2, ..., \pm l$ until we reach the level $1-\alpha$. We refer the reader to the original article [109] for a more detailed discussion of the model and a practical implementation using generating functions.

In our experiment, we set the sequencing accuracy $p = 0.85$ and the indel rate $q = 0.06$, as PB reads tend to have higher indel rates than substitution error rates. Users can adjust these parameters based on their data properties. The significance level α is 0.05 and k is 9. Using Equation (2.6), we can obtain $\rho = 54$. Using Equation (2.9) and ρ as an estimation of l, we can obtain $\delta = 5$ given $\rho = 54$. Thus, when $k = 9$, seeds with inter-seed distance $D((i_1, j_1), (i_2, j_2)) \leq 54$ and diagonal shift $|d(i_1, j_1) - d(i_2, j_2)| \leq 5$ are clustered in the same group. In addition, if k-mers (i_1, j_1) and (i_2, j_2) are in the same group and k-mers (i_2, j_2) and (i_3, j_3) are in the same group, we will cluster (i_1, j_1) and (i_3, j_3) as well.

2.2.4 Group Chaining

With group hit criteria, GroupK can find short similar regions. To identify the overlapping region, we aim to find a chain of group seed matches that maximizes the number of matched bases. We used the modified sparse dynamic programming for chaining [54, 68].

After generating a chain of group seed matches, we need to determine whether this chaining defines an overlap. We develop two criteria for this purpose. First, we calculate the expected number of matched bases from the group hit criteria, assuming that the chain covers the possible overlapping region with length L_O (L_O can be estimated by the extension of both ends of the

optimal chain). We used the following equation to calculate the expected number of the matched bases n_e:

$$n_e = \frac{1}{c} \cdot \frac{L_O}{\rho} \cdot k \tag{2.10}$$

Where c is a coefficient to control the criteria, k is the size of k-mer, ρ is the group hit criteria for the inter-k-mer distance. We only report the chaining result if the number of matched bases $n \geq n_e$. Second, we require that both reads have similar sizes inside the overlapping region.

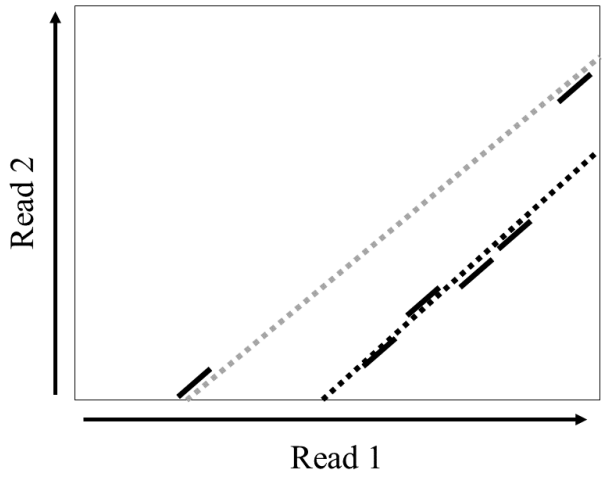

Figure 2.6: An example of the overlap size estimation. The suffix of read 1 and the prefix of read 2 form an overlap. Each short solid line represents a group seed match in the optimal chain. The black dashed line indicates the true overlap alignment region between the two reads. The gray dashed line, which is formed by the two ending group seeds in the optimal chain, can overestimate the overlap size.

In our experiment, we found that sometimes using the optimal chain generated from sparse dynamic programming may overestimate the overlap region, as shown in Figure 2.6. This overestimation can jeopardize the sensitivity of detecting small overlaps. We fix this problem by only keeping the collinear group seeds, which are used to estimate the overlap size.

2.2.5 Implementation details of the major components

2.2.5.1 Filtration by k-mer counts

In the first step of our pipeline, we use k-mer counting-based filtration to remove large numbers of read pairs that are not likely sequenced from the same loci on the underlying genome. We implemented k-mer counting using a generalized suffix array and the derived longest common prefix (LCP) array. The generalized suffix array SA is created from the concatenated reads (delimited by special characters such as \$) using a linear algorithm [125]. Then, we create the LCP using both the suffix array SA and the reversed suffix array SA' [70, 71]. Let the sequence of concatenated reads be T. Following the definition of the reversed suffix array, for a suffix starting at position $SA[i]$, we have $SA'[SA[i]] = i$. For each position i in the LCP, LCP[i] contains the size of the longest common prefix between $SA[i]$ and $SA[i-1]$. The key observation [125] for efficient computation of LCP[i] is: for a position j in T, if $LCP[SA'[j-1]]$ is L, $LCP[SA'[j]] \geq L-1$. The whole LCP array construction takes linear time to the size of T [125].

In order to count the shared k-mers between reads and also report read pairs passing the k-mer counting threshold, we use both LCP and an auxiliary data structure recording the read IDs (denoted as array $readID$). For a suffix starting at position $SA[i]$, its read ID is at $readID[i]$. The pseudocode of finding the number of shared k-mers can be found in Algorithm 1. In practice, we also count k-mers between a read and the other read's reverse complement.

2.2.5.2 Group seed match and chaining

Any pair of reads that pass the above filtration stage will be used as input for finding group seed matches. All other pairs will be discarded. Currently, we are using the codes of YASS [108, 110] for finding the group seed matches. The program uses hashing table to find short exact matches

Algorithm 1 Counting k-mers between reads

Input: LCP array $LCP[1..n]$, read ID array $readID[0..n]$, k-mer size k, k-mer counts threshold τ
Output: Read pairs whose shared k-mer $counts \geq \tau$

Initialize a map $counts[key][value]$ for recording k-mer counts
for $i = 1$ to n **do**
    $readi \leftarrow readID[i]$
    $j \leftarrow i + 1$
    $L \leftarrow LCP[j]$
    while $L \geq k$ and $j \leq n$ **do**
        $read\,j \leftarrow readID[j]$
        if $readi < read\,j$ **then**
            $key \leftarrow readi : read\,j$
        end if
        if $read\,j < readi$ **then**
            $key \leftarrow read\,j : readi$
        end if
        if $counts[key]$ does not exist **then**
            $counts[key] \leftarrow 1$
        else
            $counts[key]$++
        end if
        $j++$
        $L \leftarrow \min(L, LCP[j])$ ▷ $\min(L, LCP[j])$ returns the minimum of L and $LCP[j]$
    end while
end for
for all key in $counts$ **do**
    if $counts[key] \geq \tau$ **then**
        output key ▷ key contains read pair IDs passing the k-mer count threshold
    end if
end for

and creates the groups of matches on the fly. It is our future work to re-implement group seed matching using more efficient indexing-based methods. Implementation of chaining algorithm has been modified from the program of global chaining algorithm in SeqAn library [32].

2.2.5.3 Time complexity analysis

If we have N reads with average read length L, our text size is NL. Therefore, we have NL elements in the suffix array and the corresponding LCP array. For each suffix in the suffix array, suppose on average, it can form LCPs with m other suffixes with size above k, which is the size of k-mer used in the k-mer-count filtration steps. So the time complexity of finding all the shared k-mers for all possible reading pairs is $\mathcal{O}(mNL)$. If k is large enough (e.g., $k = 15$ and 11 in our experiment), we have $m \ll NL$, so the time complexity will be dominated by NL.

For N' read pairs that pass the filtration stage, let the average number of k-mer hits for each pair be q. Sorting the hits will need $\mathcal{O}(q \log q)$ and iterating through all hits to find groups is linear to q. For all the read pairs, the time complexity is $\mathcal{O}(N'q \log q)$.

Assuming finally we have r group seeds, the chaining procedure has complexity in $\mathcal{O}(r \log r)$ for each read pairs. For all the read pairs, the time complexity is $\mathcal{O}(N'r \log r)$. It is practically very fast because the number of group seed matches is very small compared to the original seed hits (indicated by Figure 2.3).

2.3 Results

We focus on evaluating the sensitivity and precision of overlap detection. We applied GroupK to three PB datasets and one ONT dataset: a simulated PB RSII *E. coli* sequencing dataset, a real PB RSII *E. coli* sequencing dataset, a PB RSII human foot metagenomic sequencing dataset, and an

ONT *E. coli* (SQK-MAP-006) dataset. For simulated *E. coli* dataset, we have the true sampling position for each read as our ground truth. For the real *E. coli* dataset and human foot dataset, we determine the ground truth via BLASR's [21] alignments against the reference genome.

We benchmarked GroupK's performance with Minimap [87], Minimap2 [88], DALIGNER [102], MHAP [12], and GraphMap [137]. Those tools and methods are representative overlap detection tools for long erroneous reads from PB or ONT [26]. All the detailed parameters can be found at the website listed in [33]. Our main metrics include: (1) sensitivity, which measures the ratio of the true overlaps identified by each program to the whole set of overlapping pairs; (2) precision, which quantifies the ratio of true overlap detected by each program to the total reported overlapping pairs; and (3) F1 score, which is the harmonic mean of sensitivity and precision. A reported overlapping pair is regarded as correct if it is also present in our ground truth. The detailed overlap region and overlap length were not considered in the current evaluation because these read pairs can go through a more accurate alignment program for generating the final overlap alignment. As we discussed before, our goal is to identify overlapping reads without using error correction. All tested tools will thus be applied to their raw data set.

2.3.1 Simulated *E. coli* dataset

We first evaluated the performance of our method on a simulated *E. coli* dataset. The dataset was generated using PBSIM [114] with *E. coli* K-12 MG1655 as the reference genome [60]. The length distribution and the quality profile were derived from real PB P6-C4 *E. coli* dataset [118]. The simulated dataset has 5,620 reads, with average length of 8,344.78 bps and 14.5% average error rate (8.6% insertions, 4.4% deletions, and 1.4% substitutions).

From the report by PBSIM, we can obtain the exact locations where the simulated reads are sampled in the genome. This information provides us with the ground truth for reads' overlaps so

that we can calculate the sensitivity and precision.

Following the pipeline we discussed in Section 2.2, we first used k-mer-counting as the filtration stage. According to Equation (2.1), Equation (2.3), and Figure 2.5, we discarded all read pairs with less than two 15-mer matches. The sensitivity of the filtration is 0.979 and only about 6% of read pairs are kept for downstream analysis.

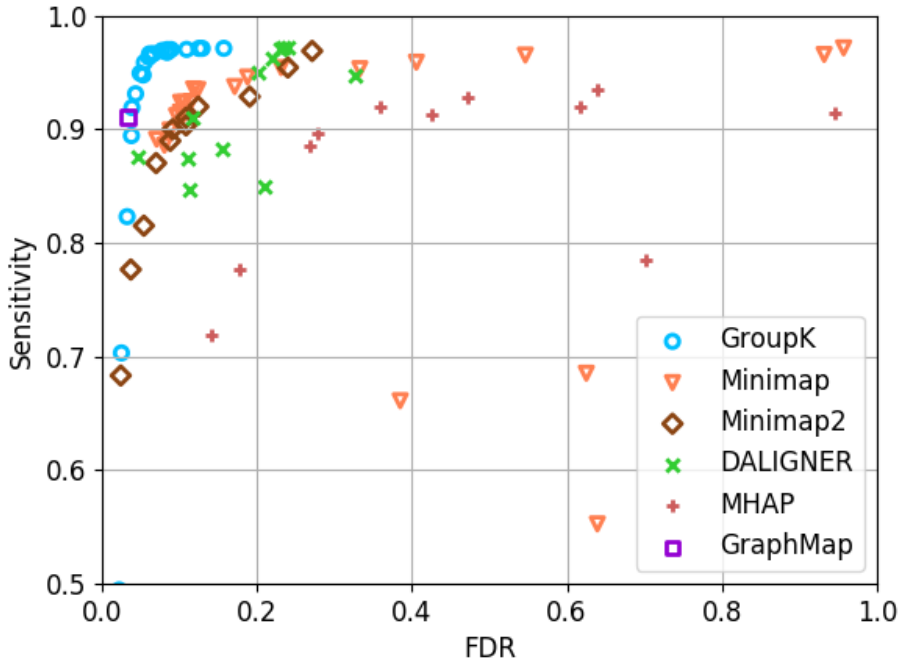

Figure 2.7: The ROC-like plot using GroupK, Minimap, Minimap2, DALIGNER, MHAP, and GraphMap on the simulated PB *E. coli* dataset. The x-axis represents the false discovery rate (FDR $= 1 -$ precision). Y-axis is the sensitivity (0.5 to 1).

We evaluated the performance of our tool by adjusting the group seed match criteria coefficient c, which is introduced in Section 2.2. With the increase of c, sensitivity will become higher, and the precision will become lower. As shown in Figure 2.7, GroupK can achieve 5% to 6% improvement on the sensitivity with similar precision to other overlap detection tools.

	GroupK	Minimap	Minimap2	DALIGNER	GraphMap	MHAP
Time (seconds)	1871	30	16	39	171	858
Memory (GB)	1.994	1.754	1.097	2.288	1.873	2.562

Table 2.1: Computational performance on the simulated *E. coli* dataset. The computational performance of overlap detection using GroupK, Minimap, Minimap2, DALIGNER, MHAP, and GraphMap on the simulated *E. coli* dataset.

2.3.1.1 Running time and memory usage

We evaluated the running time and the peak memory usage of the tested tools in this experiment. We run all overlap detection tools with a single core of 2.4Ghz 14-core Intel Xeon E5-2680v4 CPU and 32 GB memory requested from the High-Performance Computing Center at Michigan State University. The performance is measured with the best F1 score. The results are reported in Table 2.1. For the memory usage, Minimap2 is the most efficient one but all others are comparable. GroupK is slower than other tools, partially because we use small k-mers. We found that the bottleneck of our program is the group matching stage, which accounts for about 1200 of 1871 seconds. By implementing a more efficient indexing-based method, we expect to reduce the running time of this stage. For example, we can speed up k-mer counting by adopting the method used in KMC [77].

2.3.2 Real PB *E. coli* dataset

After using the simulated dataset to evaluate our method's performance, we applied GroupK to a real PB RS II (P6-C4) *E. coli* dataset [118]. The coverage of the whole dataset is 150X. To test the performance of low coverage data, we sampled a 15X coverage dataset based on the read length distribution of the whole dataset. The dataset has 14,262 reads, with the average length of 4,882.09 bps and average error rate of 14.14% (error rate is estimated using quality score).

We applied BLASR to map the reads to the reference genome to estimate the ground truth

(BLASR was run with parameters: minReadLength, 2000; maxScore, 1000; maxLCPLength, 16; minMatch, 12; m 4 and nCandidates/bestn set to 10× sequencing coverage). The mapping result from BLASR may contain short alignments due to the repeat in the genomes. So when we determine the ground truth, we only consider the alignments that cover at least 80% of the read. By removing short noisy alignments, we can make sure the BLASR alignments are close to the underlying ground truth.

	GroupK	Minimap	Minimap2	DALIGNER	GraphMap	MHAP
Best F1 score	**0.9311**	0.9037	0.8426	0.8340	0.7758	0.7023
Sensitivity	**0.9330**	0.8939	0.8778	0.9066	0.6741	0.7258
Precision	**0.9292**	**0.9138**	0.8101	0.7722	**0.9137**	0.6802

Table 2.2: Overlap detection on the real *E. coli* dataset. The performance of overlap detection using GroupK, Minimap, Minimap2, DALIGNER, MHAP, and GraphMap on the real PB RS II (P6-C4) *E. coli* dataset. Here we only report the experiment results with the highest F1 score for each tool.

We used the same filtration setup adopted in the simulated *E. coli* experiment. Among all overlap detection tools we tested, GroupK still achieved the highest sensitivity with comparable precision (Table 2.2). With slightly higher precision, our sensitivity is 4% better than the next best tool, Minimap. Compared to the previous experiment, the difference in sensitivity is smaller. One reason lies in the construction of the ground truth dataset. In the simulated dataset, we used the sample positions of all reads to determine whether two reads form an overlap. Thus, that dataset can include reads with small overlaps or reads with higher error rates. In this dataset, our method discarded BLASR alignments with high error rates and the remaining alignments have higher similarities with the reference genome and thus produce fewer "hard cases". As Minimap is the second best tool for this dataset, we further analyzed the performance of GroupK and Minimap on read pairs of different overlap size in the next section.

2.3.2.1 Performance with different overlap size

We divide all overlapping pairs into bins of width 500 based on the overlap size. For example, the first bin has the read pairs with overlap sizes from 0 to 499, and the second bin has read pairs with overlap sizes from 500 to 999, and so on. For each bin, we compared the sensitivity of GroupK and Minimap using parameters yielding the similar precision in Figure 2.8. For Minimap, we showed the result with the highest F1 score. For GroupK, we selected a parameter so that it achieves similar precision to Minimap (F1 score: 0.9241, sensitivity: 0.9344, precision: 0.9140). According to Figure 2.8, GroupK has much better sensitivity when the overlap size is less than 2,000. As we showed in Figure 2.1, there are a significant number of overlaps with overlap size smaller than 2,000 even for 30X coverage. Being able to identify small overlaps allows us to generate more complete assemblies using long reads. This is particularly useful for low coverage data, such as what we usually have in metagenomic datasets.

2.3.3 Human Foot Metagenomic Dataset

One of the major utilities of our tool is to identify overlaps between reads in metagenomic data that are sequenced using the PB platform. For complicated microbial communities, metagenomic data containing only short reads poses serious computational challenges for *de novo*assembly and the downstream composition/functional analysis. Long reads hold the promise to produce more complete and accurate microbial genome assemblies for the metagenomic dataset. In this experiment, we evaluated the performance of overlap detection for a mock metagenomic dataset constructed from a real human foot dataset [144]. A particular challenge for this experiment is the low coverage of the component species in the metagenomic dataset, which could be caused by sequencing throughput and complexity of the sample.

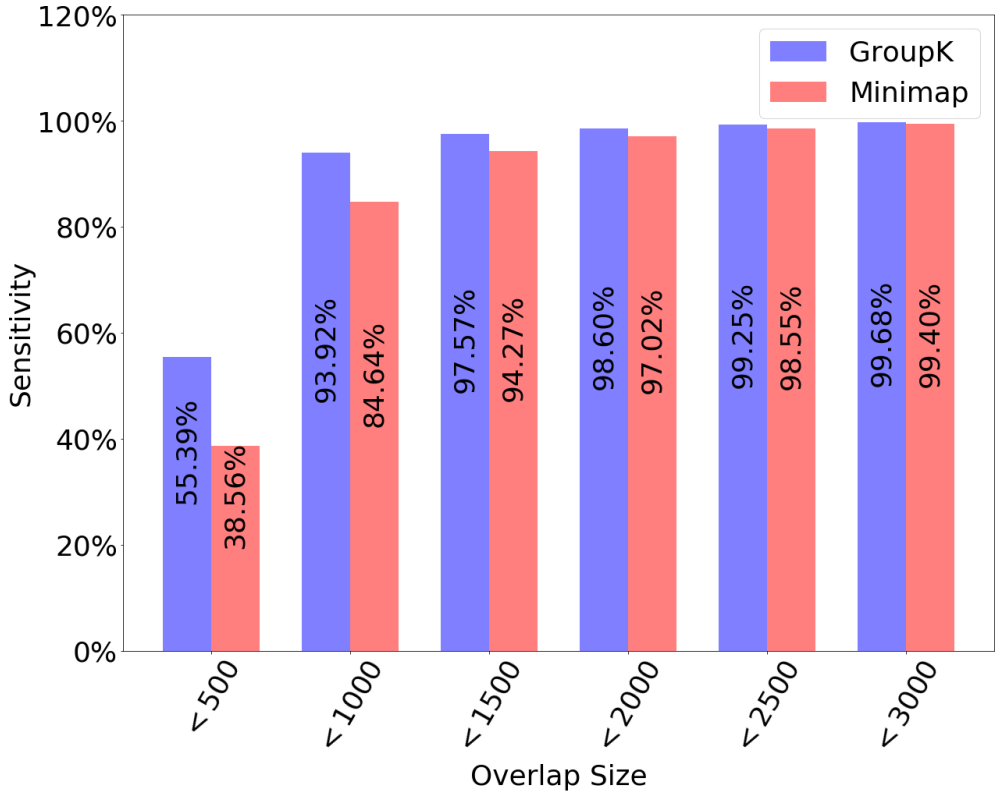

Figure 2.8: Sensitivity of GroupK and Minimap for detecting overlaps of different size on PB *E. coli* dataset. The x-axis represents the overlap size. Y-axis is the corresponding sensitivity of the bin. The X-axis bin width is 500 and the figure only included the first 6 bins (i.e. up to overlap size 3000) as their sensitivity becomes more similar with the increase of the overlap size.

The human foot sample was sequenced by linear PB RSII TdT (terminal deoxynucleotidyl transferase). Sequences that can be mapped to the human genome were removed as host-derived DNA. According to the Supplementary Materials of [144], there are about 1,000 bacteria and viruses in this metagenomic dataset. However, we cannot evaluate the performance of overlap detection on all the reads from the 1,000 microbes because the coverages of many species are too low to yield meaningful overlaps. In order to construct the ground truth, we need to align the reads against species with known reference genomes and reasonable coverage. Thus, we only choose reads satisfying the following criteria: 1) the reads are sequenced from a species with known reference genome; and 2) the coverage of the species cannot be too small (e.g., >3X coverage). Based on these criteria, we keep the reads sequenced from three bacteria: *Corynebacterium aurimucosum* (6.3X Coverage), *Corynebacterium tuberculostearicum* (8.5X Coverage), and *Staphylococcus hominis* (3.2X Coverage). The reads are recruited via BLASR. The alignment positions are used to determine which reads form an overlap. Note that *Corynebacterium aurimucosum* and *Corynebacterium tuberculostearicum* belong to the same genus and may contribute to the false positive overlap detection due to their shared regions. The average length of the reads is 1696.25 bps, which is much shorter than the reads in the previous experiments.

As this dataset contains much shorter reads, the expected number of k-mer hits will change. Intuitively we need to use shorter k-mers to ensure high filtration sensitivity. Using the read length distribution, we calculated $E[X_r]$ (Equation 2.1) and $E[X_o]$ (Equation 2.3) and determined the k-mer counting-based filtration criteria. In this experiment, we only kept read pairs that share at least three 11-mers.

For this mock metagenomic dataset, GroupK yielded significantly better performance than other tools on metrics including F1 score, sensitivity, and precision (Table 2.3). Compared to other tools, GroupK can produce much higher sensitivity without sacrificing precision, leading to

	GroupK	Minimap	Minimap2	DALIGNER	GraphMap	MHAP
Total:						
Best F1 score	**0.9163**	0.8306	0.8776	0.6911	0.7188	0.7812
Sensitivity	**0.8954**	0.7802	0.8352	0.6012	0.5721	0.6803
Precision	0.9381	0.8880	0.9245	0.8027	**0.9666**	0.9174
C. aurimucosum:						
Best F1 score	**0.9512**	0.8072	0.9117	0.7432	0.7397	0.8545
Sensitivity	**0.9228**	0.6858	0.8467	0.6266	0.5892	0.8045
Precision	**0.9814**	**0.9806**	**0.9874**	0.9131	**0.9937**	0.9111
C. tuberculostearicum:						
Best F1 score	**0.9454**	0.8688	0.9050	0.8315	0.7276	0.7961
Sensitivity	**0.9105**	0.7958	0.8346	0.7150	0.5727	0.7355
Precision	**0.9830**	0.9567	**0.9884**	**0.9934**	**0.9977**	0.8675
S. hominis:						
Best F1 score	**0.9163**	0.7651	0.8024	0.8754	0.6343	0.6861
Sensitivity	**0.8733**	0.6887	0.6813	**0.7967**	0.4658	0.6348
Precision	0.9245	0.8606	0.9759	0.9713	**0.9938**	0.7464

Table 2.3: Overlap detection on the human metagenomic dataset. The performance of overlap detection using GroupK, Minimap, Minimap2, DALIGNER, MHAP, and GraphMap on the mock metagenomic dataset. Here we only report the experimental results with the highest F1 score for each tool.

the higher F1 score. Besides evaluating the performance of various tools on all the reads from the three species, we also reported the performance of different overlap detection tools on each single bacteria dataset without mixing with other species (Table 2.3). In these tools, GraphMap has high specificity for all three with sacrifice of sensitivity. However, GroupK still achieves the best performance overall. The comparisons suggest that our method has great potential to detect overlaps for data with very low coverage (around 5X). This will enable better assembly for PB sequenced metagenomic data, which will become more available with the advances of long read sequencing technologies.

2.3.4 Real ONT *E. coli* dataset

We also tested our method on one ONT dataset. We used downsampled 15X coverage 2D reads from the SQK-MAP-006 dataset as 2D reads provide higher quality than 1D reads. We followed the same pipelines we used for the real *E. coli* PB dataset. As 2D ONT reads have similar error rates to the PB reads, we expect that our tool can still achieve reasonable performance given the same setup for the PB dataset. Therefore, we used the same parameters as the ones we used for the PB *E. coli* dataset.

	GroupK	Minimap	Minimap2	DALIGNER	GraphMap	MHAP
F1 score	**0.9383**	**0.9310**	0.9090	0.8272	**0.9280**	0.8122
Sensitivity	**0.9597**	**0.9546**	0.9362	0.8730	0.8991	0.8871
Precision	0.9178	**0**.9085	0.8833	0.7860	**0.9589**	0.7490

Table 2.4: Overlap detection on the ONT *E. coli* dataset. The performance of overlap detection using GroupK, Minimap, Minimap2, DALIGNER, MHAP, and GraphMap on the real ONT SQK-MAP-006 *E. coli* dataset. Minimap2 uses the ava-ont setup, which is optimized for ONT data.

Among these tools, GraphMap was designed for ONT data, Minimap2 provides a specific setup for finding the overlap on ONT dataset. All other tools are not specifically designed for ONT datasets. GroupK achieves the best F1 score compared to other tools' default setup while keeping the highest sensitivity (Table 2.4). This result suggests that our strategy is robust with different types of long reads.

2.4 Discussion

Seeding is a key step for overlap detection because of the high error rate of long reads. Successful seeding strategies should balance the sensitivity and the specificity to achieve the optimal performance. Popular seeding methods include maximal exact matches, spaced seeds, and gapped spaced

seeds. However, to successfully find a hit between two reads, these methods still need either to find relatively long continuous exact matches (large k-mer) or to find inexact matches following certain error patterns (spaced seed). Compared to these methods, group seed matching is more flexible as it requires multiple short exact matches without specifying the error patterns. This flexibility leads to high sensitivity, and meanwhile the specificity is still guaranteed with the group seed match criteria.

Currently the group seed matching step based on hash table is the bottleneck of our overlap detection pipeline. A new method that can improve the running time efficiency of this step is needed to make the algorithm achieve the same speed as other faster overlap detection tools.

2.5 Conclusions

In this work, we developed an overlap detection tool for third-generation sequencing data. By adopting the group hit criteria to cluster a group of short k-mer hits that satisfy statistically derived distance constrains, our method can improve the sensitivity of overlap detection without sacrificing precision. Our experimental results have shown that for datasets with low sequencing coverage, our program can detect significantly more overlapping pairs while keeping high precision. One utility of our approach is to detect small overlaps between long reads of rare species in a microbial community.

Chapter 3

profile HMM model and homology search [1]

3.1 Introduction

Compared to Illumina, the representative secondary generation sequencing platform, the major disadvantages of PB include high sequencing error rate (11-15%), lower throughput, and higher cost per base [124,126]. Similar to pyrosequencing data, most of the errors are insertion or deletion errors. The high error rate poses challenges for all downstream sequence analysis. In particular, during homology search for genome annotation, sequences are aligned to characterized protein sequences or families. Insertion or deletion errors in genes will cause frameshifts and may only lead to marginal alignment scores and short alignments [154]. As a result, it is hard to distinguish true alignments from random alignments. The inaccurate homology search results can incur errors in structural and functional annotation.

Different strategies have been proposed or implemented to avoid or correct sequencing errors in PB data. There are various PB sequencing projects that mainly use circular consensus sequencing (CCS) reads with sufficient sequencing passes. A coverage of 15 passes yields $> 99\%$ accuracy [128]. However, CCS reads are much shorter than the continuous long reads of PB data. In addition, the amount of CCS reads is much less than all the output of PB data. Thus, using only CCS reads does not take full advantage of the sequencing power and strength of PB data.

[1] Du, Nan, and Yanni Sun. "Improve homology search sensitivity of PacBio data by correcting frameshifts." Bioinformatics 32.17 (2016): i529-i537.

One popular strategy to handle sequencing errors of PB data is based on hybrid sequencing [78]. As Illumina produces many more accurate but shorter reads, methods are developed to correct errors by aligning short reads to long PB reads. Yet, this method needs preparation of at least two sequencing libraries and several types of sequencing runs, which is not cost-effective for many applications.

Unlike hybrid sequencing, there are methods that do not require highly accurate short reads for error correction. One representative method is described in Chin et al.'s hierarchical genome-assembly process HGAP [24], which aligns short sequences to the longest reads of the same sequencing library of PB. As the sequencing errors in PB reads occur randomly, the inferred consensus sequence from the alignment between the short reads and long reads represent the high-quality sequence. Despite its success, there is still room to improve the error correction performance for the consensus sequence extraction stage in HGAP. In particular, its performance is heavily affected by the coverage of the aligned short sequences against the long seed sequences. The regions with more short sequences aligned have better error correction performance than other regions.

After error correction, corrected PB reads can usually achieve more sensitive homology search results compared to the raw data. In particular, when the coverage is high, the corrected reads from HGAP can achieve alignment scores similar to the ground truth. However, in practice, not all PB sequencing projects can have sufficient coverage for all regions, transcripts, or genomes. For example, HGAP failed to assemble the data from the arm sample in human skin microbial community [144] because of low coverage of the data set. Figure 3.7 in our experimental results show that the difference of the alignments' scores, lengths, and E-values between HGAP's corrected reads and the ground truth is still significant. Thus, there is still a need for homology search tools designed for PB data.

In this work, we designed and implemented Frame-Pro, a homology search tool for PB reads.

The experimental results showed that our tool can significantly improve the homology search sensitivity while also correcting sequencing errors. Our method incorporated two key observations. First, as shown by HGAP, sequencing errors in PB are distributed randomly and thus the consensus sequences tend to be closer to ground truth. Our work incorporated this method. Second, we identify frameshifts caused by sequencing errors using characterized protein families as the guidance. Essentially our method corrects errors by maximizing both alignment score against protein families and local coverage score in a constructed alignment graph. Both observations are used together to boost the performance of both homology search and error correction.

The remainder of this Chapter is organized as follows. Section 3.2 briefly reviews other frameshift error detection tools and their limitations in protein domain classification in PB data sets. Section 3.4 describes the dynamic programming algorithm that incorporates consensus sequence finding and Viterbi algorithm for error correction and sequence alignment. In Section 3.5, we demonstrate the results of error correction and homology search by applying our tool to simulated and real PB data. We also benchmark our tool with HGAP, a successful error correction method without relying on hybrid sequencing. Finally, Section 3.6 concludes and suggests directions for future work.

3.2 Related work

3.2.1 Profile homology search

Homology search is still an important step in sequence-based functional analysis for genomic data. By comparing query sequences against reference sequences or profiles, i.e., a family of homologous reference sequences, functions and structures can be inferred. The representative tools for sequence homology search and profile homology search are BLAST [4] and HMMER [38], re-

spectively. Profile homology search has several advantages over pairwise alignment tools such as BLAST. First, the number of gene families is significantly smaller than the number of sequences, rendering much faster search time. For example, there are only about 13,000 manually curated protein families in Pfam, but these cover nearly 80% of the UniProt Knowledgebase and the coverage is increasing every year as enough information becomes available to form new families [45]. The newest version of HMMER [38] is more sensitive than BLAST and is about 10% faster.

Second, previous work [35] has demonstrated that using family information can improve the sensitivity of a remote protein homology search, which is very important for various sequencing data such as metagenomic data analysis. These data sets may contain species remotely related to ones in the reference database and require sensitive homology search. Thus, in this work, we focus on implementing profile homology search for PB data. The method can be extended to pairwise sequence alignment. As HMMER is the most widely used profile alignment tool, we focus on evaluating the alignment performance using HMMER.

The protein domains families used in our experiments are downloaded from Pfam. Other databases such as TIGRFAMs [57], FIGfams [98], InterProScan [152], and FOAM [123] can be used too as long as profile hidden Markov models can be trained.

3.2.2 Related work on frameshift correction

Usually, when comparing a DNA sequence with a protein sequence or family, six-frame translations are conducted and one of the reading frame should lead to statistically significant alignment if the query and the reference are homologous. However, frameshifts caused by insertion or deletion errors make the correct translation consist of alternating reading frames. Without knowing the error positions, choosing the correct frames for each fragment between errors is challenging.

A number of programs exist to handle frameshifts through DNA vs. protein sequence align-

ment. Simple methods such as BLASTX discard sequences that might contain frameshifts rather than trying to fix them. Other tools [17, 22, 49, 50, 53, 59, 120, 121, 155] are available to detect and fix frameshift errors automatically. Besides detecting frameshift in sequence alignment, some programs [5, 14, 76, 131] focus on frameshift detection during gene finding and use *ab initio* methods. These programs are not designed for PB data. In addition, they cannot be applied to protein profile homology search.

Alternatively, Genewise [13], a widely used DNA vs. protein alignment tool allows comparison of a DNA sequence with a protein family. But it does not consider the sequencing error properties of NGS data. The most relevant works to ours include [154], which modified Viterbi algorithm to improve homology search for pyrosequencing data. But it does not have satisfactory performance for PB data because the sequencing error properties of pyrosequencing and PB are different. Pyrosequencing reads have lower error rate and most of the errors are located inside homopolymer regions. These properties make error correction easier than PB, which have higher error rates and the errors can occur more randomly. FrameBot [150] is another relevant work for correcting frameshifts caused by sequencing errors. But it is not designed for profile homology search. And, like other tools, it is only optimized and tested on sequencing data with lower error rate than PB.

HGAP [24] is another highly relevant work because it contains error correction stage for PB data. As discussed in Section 3.1, its performance is heavily affected by sequencing coverage.

3.3 Profile hidden Markov model

Before introducing the augmented Viterbi algorithm that used in Frame-Pro, we first reviewed the conceptes and algorithms of profile hidden Markov model. The content of this section is mainly

adapted from [35, 36].

3.3.1 Markov chains

We start from Markov chain, which is the simplest probabilistic model of sequence. In the model, the probability of a symbol in the sequence depends on the previous symbol. In Markov chain, each symbol (e.g., residue of DNA) corresponds to a state in the chain.

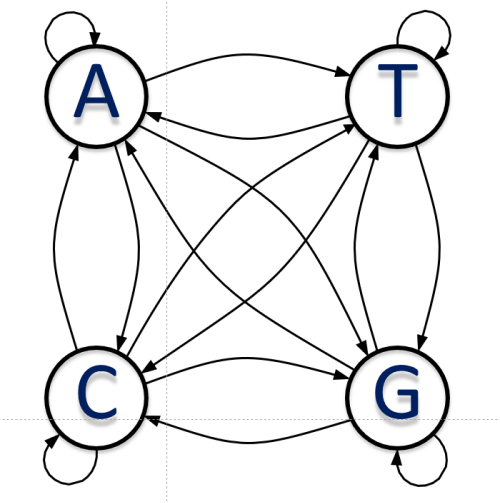

Figure 3.1: A Markov chain example of DNA. There are four states for each of nucleotides `A`, `C`, `G` and `T`. A transition probability is associated with each arrow in the figure.

The transition probabilities a_{st} is the probability that a certain state following another state:

$$a_{st} = P(x_i = t | x_{i-1} = s) \tag{3.1}$$

The probability of the sequence can be write as

$$\begin{aligned} P(x) &= P(x_L, x_{L-1}, ..., x_1) \\ &= P(x_L | x_{L-1}, ..., x_1) P(x_{L-1} | x_{L-2}, ..., x_1) ... P(x_1) \end{aligned} \tag{3.2}$$

by repeat applying$P(X,Y) = P(X|Y)P(Y)$. In Markov chain, the probability of each symbol x_i depends only on the previous symbol x_{i-1}, not on entire previous sequence. So the previous equation can be rewrite as

$$\begin{aligned} P(x) &= P(x_L|x_{L-1})P(x_{L-1}|x_{L-2})...P(x_1) \\ &= P(x_1)\prod_{i=2}^{L} a_{x_{i-1}x_i} \end{aligned} \tag{3.3}$$

This is the general equation for the probability of a specific sequence of any Markov chain.

3.3.2 Hidden Markov models

In the Markov chain, there is a one-to-one correspondence between the states and the symbols. Since we can directly observe the symbols in the sequences (e.g., residues in DNA sequences), we can also confirm the state from the observation. However, in the hidden Markov model, one state may emit many different symbols. So from the observation of symbols, we cannot confirm the state from our observation. In other words, the state is hidden now, so we call such model hidden Markov model.

We fomulate the hidden Markov model as following: We call the sequence of the states *path*, π. The path itself is a simple Markov chain, the probability of a state in the path only depends on the previous state. The *i*th state in the path is called π_i. The transition probability of the chain is:

$$a_{kl} = P(\pi_i = k|\pi_{i-1} = k) \tag{3.4}$$

Because we have decoupled the symbols b from the states k, we must need a new set of parameters, emission probabilities $e_k(b)$, to model the probability that symbol b is observed when in

state k:

$$e_k(b) = P(x_i = b|\pi_i = k) \tag{3.5}$$

We can write the probability of an observed sequence x and a state sequence π as:

$$P(x,\pi) = a_{0\pi_1}\prod_{i=1}^{L} e_{\pi_i}(x_i)a_{\pi_i\pi_{i+1}} \tag{3.6}$$

where we require $\pi_{L+1} = 0$. The Equation 3.6 is the HMM analogue of Equation 3.3.

In real application of HMM, there are three fundamental problems we care about [69]: (1) Given an HMM model with parameters a_{kl} and $e_k(b)$, and an observation sequence x, determine the likelihood of $P(x)$ (**likelihood** problem); (2) Given an HMM model with parameters a_{kl} and $e_k(b)$, and an observation sequence x, decode the most probable state path π^* (**decoding** problem); (3) Given an observation sequence x and states of HMM, learn the HMM parameters a_{kl} and $e_k(b)$ (**learning** problem). The Viterbi algorithm, Forward algorithm, and Forward-Backward algorithm were developed to solve those problems, respectively. We will introduce these algorithms for profile hidden Markov model in the following sections.

3.3.3 Profile hidden Markov models (profile HMMs)

Profile HMMs is a particular HMM developed to model multiple sequence alignments [82]. Here we will use profile HMMs designed for protein family multiple alignments as our example. In profile HMMs, for each position of state i in the path π, there are three possible states, match state M_i, insertion state I_i, and deletion state D_i (Figure 3.2). Insertion state I_i is used to match insertions after the residue matching the ith column of the multiple alignments. There are transitions from M_i to I_i, a loop transition from I_i to itself to accommodate multi-residue insertions, and a transition

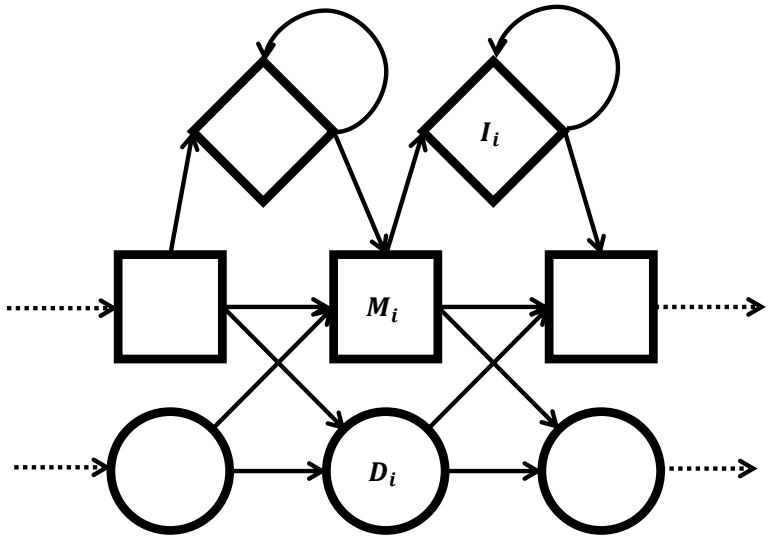

Figure 3.2: The transition structure of a profile HMM. We use squres to indicate match states, diamonds to indicate insertion states, and circles to indicate deletion states.

from I_i back to M_i. Deletion state D_i is used to handle the jump transition between non-neighboring match states without any emission. The sum of the costs of an transition M to D followed by a number of D to D transitions, then a D to M transitions will be the cost of a deletion.

3.3.3.1 Viterbi algorithm

Viterbi algorithm [148] is the most common algorithm for finding the most probable state path:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} P(x, \pi) \tag{3.7}$$

The most probable path π can be found recursively. For profile HMMs, let $V_j^M(i)$ be the maximum log-odds score of aligning a best sub-sequence $x_{1,2,..,i}$ to the HMM path up to state j, ending with x_i being emitted by state M_j, $V_j^I(i)$ be the score of the best path ending with x_i being emitted by

state I_j similarly, and $V_j^D(i)$ for the best path ending in state D_j. q_{x_i} is the emission probability of a standard random model (background distribution). Then the recursive Viterbi equations can be write as:

$$
\begin{aligned}
V_j^M(i) &= \log \frac{e_{M_j}(x_i)}{q_{x_i}} + \max \begin{cases} V_{j-1}^M(i-1) + \log a_{M_{j-1},M_j} \\ V_{j-1}^I(i-1) + \log a_{I_{j-1},M_j} \\ V_{j-1}^D(i-1) + \log a_{D_{j-1},M_j} \end{cases} \\
V_j^I(x_i) &= \log \frac{e_{I_j}(x_i)}{q_{x_i}} + \max \begin{cases} V_j^M(i-1) + \log a_{M_j,I_j} \\ V_j^I(i-1) + \log a_{I_j,I_j} \end{cases} \\
V_j^D(x_i) &= \max \begin{cases} V_{j-1}^M(i) + \log a_{M_{j-1},D_j} \\ V_{j-1}^D(i) + \log a_{D_{j-1},D_j} \end{cases}
\end{aligned}
\tag{3.8}
$$

In a common situation, the emission score $\log \frac{e_{I_j}(x_i)}{q_{x_i}}$ for $V_j^I(x_i)$ will be canceled because we assume the emission distribution $e_{I_j}(x_i)$ from the insertion states I_j is the same as the background distribution q_{x_i}. The whole process of Viterbi algorithm is described in Algorithm 2.

Algorithm 2 Viterbi algorithm

Input: DNA sequence x with length L, profile HMM with length n
Output: Most probable state path π^* with log-odds score $\log P(x, \pi^*)$

$V_0^M(0) = 0$, $V_j^S(0) = -\texttt{inf}$ for$0 < j$ and $S \in \{\text{M, I, D}\}$. ▷ Initialization
for $i = 1$ to L **do** ▷ Recursion
    $V_j^S(i) \leftarrow$ Equation 3.8
    $\text{ptr}_i(j) \leftarrow \text{argmax}_{S_{j-1}}(V_j^S(i))$
end for
$\log P(x, \pi^*) \leftarrow V_n^M(L)$ ▷ Termination
$\pi_L^* \leftarrow \text{argmax}_{n-1} V_n^M(L)$
for $i = L$ to 1 **do**
    $\pi_{i-1}^* \leftarrow \text{ptr}_i(\pi_i^*)$ ▷ Traceback
end for

3.3.3.2 Forward algorithm

To evaluate the likelihood of a sequence x, we need sum the probabilities of all possible state path to obtain the probability of x:

$$P(x) = \sum_{\pi} P(x, \pi) \tag{3.9}$$

A dynamic programming algorithm can be used to calculate this full probability recursively. This is called the forward algorithm.

We used the similar notation that used in Viterbi algorithm. The forward log-odds score $F_j^M(i)$, $F_j^I(i)$, and $F_j^D(i)$ are defined corresponding to $V_j^M(i)$, $V_j^I(i)$, and $V_j^D(i)$. So the recurrence equations of forward algorithm are:

$$\begin{aligned}
F_j^M(i) &= \log \frac{e_{M_j}(x_i)}{q_{x_i}} + \log[a_{M_{j-1},M_j} \exp(F_{j-1}^M(i-1)) \\
&\quad + a_{I_{j-1},M_j} \exp(F_{j-1}^I(i-1)) + a_{D_{j-1},M_j} \exp(F_{j-1}^D(i-1))] \\
F_j^I(x_i) &= \log \frac{e_{I_j}(x_i)}{q_{x_i}} + \log[a_{M_j,I_j} \exp(F_j^M(i-1)) \\
&\quad + a_{I_j,I_j} \exp(F_j^I(i-1))] \\
F_j^D(x_i) &= \log[a_{M_{j-1},D_j} \exp(F_{j-1}^M(i)) + a_{D_{j-1},D_j} \exp(F_{j-1}^D(i))]
\end{aligned} \tag{3.10}$$

The initialization of the forward algorithm requires that $F_0^M(0) = 0$, which is similar to Viterbi algorithm. Since we only need the probabilities of x so there is no traceback in the forward algorithm.

3.3.3.3 From multiple sequence alignment to profile HMM

Usually, we want to build profile HMMs from a multiple sequence alignment with multiple columns. We first need to determine which multiple alignment columns to assign to match states, and which to assign insertion states (Figure 3.3). We often use the heuristic rule to ignore the column for which the fraction of gap characters is greater than or equal to a column removal threshold θ [29]. Usually, the column removal threshold θ equals to 0.5. After this, we can construct profile HMMs as all the states are determined.

```
HBA_HUMAN    ...VGA--HAGEY...
HBB_HUMAN    ...V----NVDEV...
MYG_PHYCA    ...VEA--DVAGH...
GLB3_CHITP   ...VKG------D...
GLB5_PETMA   ...VYS--TYETS...
LGB2_LUPLU   ...FNA--NIPKH...
GLB1_GLYDI   ...IAGADNGAGV...
                  ***  *****
```

Figure 3.3: Ten columns from the multiple sequence alignments of seven globin proteins. The starred columns are ones that will be treated 'matches' in the profile HMM. Reprinted from [35].

Then we need to estimate the probability parameters. From multiple sequence alignment, we can align each of the row to the profile HMM. We can directly estimate the parameters from the alignments. By counting up the number of times each transition or emission is used, the probability parameters are assigned by:

$$a_{kl} = \frac{A_{kl}}{\sum_{l'} A_{kl'}} \text{ and } e_k(a) = \frac{E_k(a)}{\sum_{a'} E_k(a)} \tag{3.11}$$

where k and l are indices over states, and a_{kl} and e_k are the transition and emission probabilities, and A_{kl} and E_k are the corresponding counts.

In many situations, we only have a limited number of sequences in the training alignments. The major challenges here is that some transitions and emissions that never seen in the training alignment will be assigned zero probability. To solve the problem, one can add pseudocounts to the observed counts. The most straightforward Laplace's rule is to add one to each count we used in Equation 3.11. A better solution is using a mixture of Dirichlet distributions as the prior. Readers are encouraged to learn more details in Chapter 5 of [35].

3.3.3.4 Forward-Backward algorithm

When the path is unknown for the training process, we are no longer able to estimate the parameter value and must iterative procedure to learn the parameters. Forward-backward algorithm is the standard algorithm for leaning the parameters of HMM. It is introduced from [9] by Leonard Baum, so it is also called Baum-Welch algorithm. It is a special case of the Expectation-Maximization (EM) algorithm. The algorithm first estimated the probabilities, then derived a better estimates. The estimation is improved iteratively until some stopping criterion is reached.

We already calculated forward probability using forward algorithm. To train HMM, we also need to calculate the posterior probability that observation x_i came from state k given the observed sequence $P(\pi_i = k|x)$ using backward algorithm. We first calculated the probability of producing the entire observed sequence with the ith symbol being produced by state k:

$$\begin{aligned} P(x, \pi = k) &= P(x_1 \ldots x_i, \pi = k) P(x_{i+1} \ldots x_L | x_1 \ldots x_i, \pi = k) \\ &= P(x_1 \ldots x_i, \pi = k) P(x_{i+1} \ldots x_L | \pi = k) \\ &= f_k(i) b_k(i) \end{aligned} \tag{3.12}$$

$$b_k(i) = P(x_{i+1}...x_L|\pi = k) \tag{3.13}$$

$b_k(i)$ can be obtained by a backward recursion start from end of the sequence.

The posterior probabilities can be calculated by straightforward conditioning,

$$P(\pi = k|x) = \frac{f_k(i)b_k(i)}{P(X)} \tag{3.14}$$

where $P(x)$ is the result of forward or backward algorithm.

Then we can calculate the expected number of times of transitions and emission, A_{kl} and E_k, using the forward probabilities and backward probabilities. The detailed derivation can be found from text books [29, 35, 69]. The expected number of times that a_{kl} is used is given by:

$$A_{kl} = \sum_j \frac{1}{P(x^j)} \sum_i f_k^j(i) a_{kl} e_l(x_{i+1}^j) b_l^j(i+1) \tag{3.15}$$

where $f_k^j(i)$ is the forward variable $f_k(i)$ calculated for sequence j, and $b_l^j(i)$ is the backward variable correspondingly. Similarly, we have the equation for expected number of times that b appears in state k:

$$E_k(b) = \sum_j \frac{1}{P(x^j)} \sum_{\{i|x_i^j=b\}} f_k^j(i) b_k^j(i) \tag{3.16}$$

where the second sum calculate over the positions i for which the symbol emitted is b.

The whole process of Baum-Welch algorithm like this:

Algorithm 3 Baum-Welch algorithm

Initialize a_{lk} and e_k.
while $\Delta >$ predefined threshold Θ **do**
    Set all the A and E to their pseudocount values.
    for each sequence $j = 1$ to n **do**
        for $i = 1, ..., L$ **do**
            $f_k(i) = e_k(x_i) \sum_h f_h(i-1) a_{ik}$
        end for ▷ forward algorithm
        for $i = L-1, ..., 1$ **do**
            $b_k(i) = \sum_l a_{kl} e_l(x_{i+1}) b_l(i+1)$
        end for ▷ backward algorithm
        Add the contribution of sequence j to A and E in Equation 3.15 and 3.16
    end for
    Calculate new a_{lk} and e_k using Equation 3.11
    Calculate the new log likelihood of the model
    $\Delta \leftarrow$ change of the log likelihood
end while

3.4 Methods

Frame-Pro is designed to improve profile homology search performance for PB data. It incorporates consensus-based error correction and a modified Viterbi algorithm for finding optimal alignment. While a standard Viterbi algorithm aligns a single sequence to a hidden Markov model (HMM), our algorithm aligns an alignment graph to an HMM. In addition, we defined a new scoring system that combines the path score from the alignment graph and the alignment score in the HMM. We first introduce the construction of the alignment graph.

3.4.1 Generate sequence alignment graph

Following HGAP, we first construct a sequence alignment graph (SAG) from PB data. The details of the graph construction can be found in the supplementary material of Chin et. al's work [24]. Here we simply summarize the major steps. Reads with a length longer than a chosen cut-off are selected as seed sequences. Other reads are then aligned to the seed sequences by BLASR [21] for

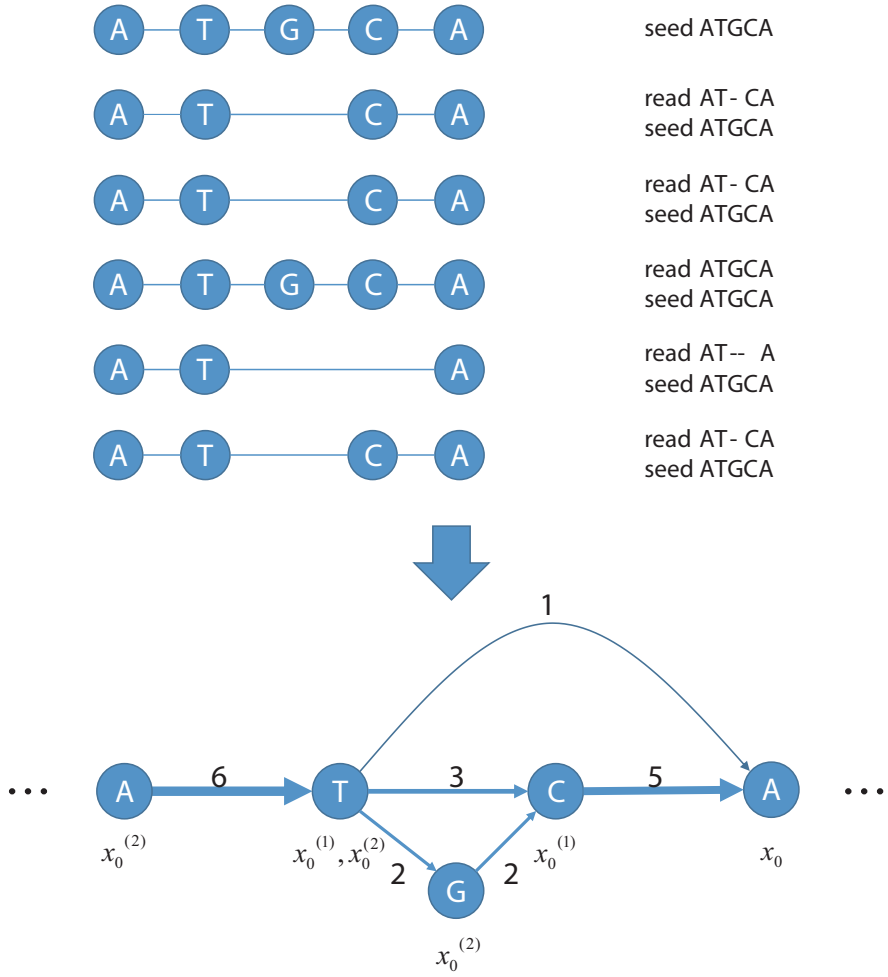

Figure 3.4: Build an example SAG from an alignment. Top panel: multiple sequence alignment of six sequences. The top sequence is the seed sequence. Bottom panel: sequence alignment graph. For node x_0, if we trace back for two more edges, we can identify three codons ending with x_0: TCA, ATA, and GCA. Each path has its specific nodes $x_0^{(1)}$ and $x_0^{(2)}$ marked. The consensus path of this part of graph is ATCA.

construction of an alignment graph (see top panel of Figure 3.4). In each aligned column, different bases are modeled as nodes for the corresponding position in an SAG. Consecutive bases are connected by an edge in the graph. The edge weight represents the number of aligned sequences that go through this edge.

The consensus sequence from the alignment graph represents the most reliable sequence. Figure 3.4 shows an example SAG and the consensus sequence. Usually, when there are sufficient reads aligning to the seed sequences, the consensus sequence can be reliably used for downstream analysis including conducting homology search. However, when the coverage is not sufficient, the chosen consensus sequence does not show significantly higher score than alternative sequences. Thus, it is difficult to extract a path that is closest to the reference sequence.

Our algorithm is much less sensitive to sequencing coverage as it uses characterized protein families as reference to choose optimal path in the alignment graph. Essentially, it aligns an alignment graph with a profile HMM, which represents a protein family. During the alignment, the algorithm chooses a sequence path in SAG and a state path in the HMM in order to maximize the combined coverage score and alignment score. Below we describe the modified Viterbi algorithm.

3.4.2 Viterbi algorithm for aligning an alignment graph with a profile HMM

Let π be a state path in a profile HMM Model M. Let $\pi^{'}$ be a sequence path in an SAG G. Our goal is to search for the optimal path pair $(\pi^*, \pi^{'*})$ such that $(\pi^*, \pi^{'*}) = \text{argmax}(\alpha S_M(\pi, \pi') + \beta S_G(\pi^{'}))$, where $S_M(\pi, \pi')$ is the alignment score between a sequence in G and the profile HMM. α and β are the weights of the HMM score and the coverage score in G, respectively. Intuitively this algorithm searches for an optimal alignment between a DNA sequence in the alignment graph G and a profile HMM M by simultaneously considering 1) the probability of a profile HMM alignment, represented by log-odds score $S_M(\pi, \pi')$, and 2) path weight in G, represented by $S_G(\pi^{'})$. To solve

the above equation, we developed the following dynamic programming algorithm, which is an augmented Viterbi algorithm [35]. The recursive equations can be extended to forward algorithm and also posterior probability calculation for an HMM.

Input: a SAG G generated by alignments between a long DNA seed sequence x_{seed} and multiple shorter sequences, a profile HMM M. Notations of M will be described below.

Output: the error-corrected DNA sequence x'_{seed} and its optimal alignment with M.

Algorithm: we first define notations that will be used in the recursive equations.

- **Notations of the profile HMM model M:** The detailed descriptions of a profile HMM model can be found in the literature [35, 36]. A profile HMM model M consists of match states M_j, deletion states D_j, and insertion states I_j for each position j, which is the index of a conserved column in a multiple sequence alignment. $a_{s_i s_j}$ is the transition probability from state s_i to s_j. As the HMM is constructed by aligned protein sequences while the graph model is constructed by aligned DNA sequences, we need to translate the sequences in G into amino acids. Here, let $T(x_{i-2}x_{i-1}x_i)$ be the amino acid translated from a codon $x_{i-2}x_{i-1}x_i$. $e_s(T)$ is the emission probability for state s to emit T. Compared to the topology of Plan 7 model used by HMMER [38], one of the major changes we made is that N and C are responsible for emitting all DNA bases that are outside of the protein domain.

- **Notations of the SAG:** Let x_i represent the ith node in the topologically sorted list of the graph. As the alignment graph is constructed in DNA space, x_i also represents a base from the aligned sequences. $x_i^{(k)}$ represents a node, from which to node x_i there exists a path consisting of k edges. According to this definition, when $k = 1$, there is an edge from $x_i^{(1)}$ to x_i. When $k = 2$, a codon is formed by three bases: $x_i^{(2)}$, $x_i^{(1)}$, and x_i. For example, in Figure 3.4, both nodes labeled with T and C are $x_0^{(1)}$. $S_G(\pi')$ represents the path score for

π', which can be a sequence of any length in G. We will define the path score following the equations.

- **Subproblems and the recursive equations:** For a sequence path π' ending at node x_i in G, and a state path π ending with index j in M, the dynamic programming algorithm intends to maximize the combined path score and alignment score: $\alpha S_G(\pi'_{1..i}) + \beta S_M(\pi'_{1..i}, \pi_{1..j})$. Depending on the ending states, we need to consider multiple cases.

  $V_j^M(x_i)$ is maximum score of aligning a sequence path ending with x_i in G to the HMM up to state M_j, under the constraint that the animo acid translated by the last three bases x_i, $x_i^{(1)}$, and $x_i^{(2)}$ in G is emitted by match state M_j. Note that there can be multiple sequence paths in G ending with x_i. So the last three bases of any such sequence path can be generally represented by $x_i^{(2)}x_i^{(1)}x_i$. And the translated amino acid is thus $T(x_i^{(2)}x_i^{(1)}x_i)$.

  $V_j^I(x_i)$ is the maximum score of aligning a sequence path ending with x_i in G to the HMM up to state V_j, under the constraint that the animo acid translated by the last three bases x_i, $x_i^{(1)}$, and $x_i^{(2)}$ in G is emitted by insertion state I_j.

  $V_j^D(x_i)$ is the maximum score of aligning a sequence path ending with x_i in G to the HMM up to state D_j. $V^N(x_i)$ is the maximum score of aligning a sequence path ending with x_i in G to the HMM up to state N, given x_i being emitted by the special state N. Similarly, $V^C(x_i)$ is the maximum score of aligning a sequence path ending with x_i in G to the HMM up to state C, given x_i being emitted by the special state C.

  For brevity of presentation, S in the following equations represents $S(x_i^{(3)}x_i^{(2)}x_i^{(1)}x_i)$, which is the path score from the possible third base of the preceding codon to the current codon ending with x_i. Note that the path score is not a simple summation of edge weights as in a standard graph. We will define the path score after the recursive equations. T is the abbrevi-

ation of $T(x_i^{(2)}x_i^{(1)}x_i)$ in following equations.

$$
\begin{aligned}
V_j^M(x_i) &= \max\begin{cases} V_{j-1}^M(x_{i^{(3)}}) + \alpha e_{M_j}(T) + \alpha \log a_{M_{j-1},M_j} + \beta S \\ V_{j-1}^I(x_{i^{(3)}}) + \alpha e_{M_j}(T) + \alpha \log a_{I_{j-1},M_j} + \beta S \\ V_{j-1}^D(x_{i^{(3)}}) + \alpha e_{M_j}(T) + \alpha \log a_{D_{j-1},M_j} + \beta S \\ V^N(x_{i^{(3)}}) + \alpha e_{M_j}(T) + \beta S \end{cases} \\
V_j^I(x_i) &= \max\begin{cases} V_j^M(x_{i^{(3)}}) + \alpha e_{I_j}(T) + \alpha \log a_{M_j,I_j} + \beta S \\ V_j^I(x_{i^{(3)}}) + \alpha e_{I_j}(T) + \alpha \log a_{I_j,I_j} + \beta S \end{cases} \\
V_j^D(x_i) &= \max\begin{cases} V_{j-1}^M(x_i) + \alpha \log a_{M_{j-1},D_j} \\ V_{j-1}^D(x_i) + \alpha \log a_{D_{j-1},D_j} \end{cases} \\
V^N(x_i) &= \max\begin{cases} V^N(x_i^{(1)}) + \beta S_{x_{i^{(1)}}x_i} \end{cases} \\
V^C(x_i) &= \max\begin{cases} V^C(x_i^{(1)}) + \beta S_{x_{i^{(1)}}x_i} \\ V^M(x_i^{(1)}) + \beta S_{x_{i^{(1)}}x_i} \end{cases}
\end{aligned}
$$

- **Calculate path scores** S**:** we followed the same equation in HGAP paper [24] to calculate a path score, which is the sum of the scores of all nodes in the path. In a SAG G, the local coverage for a position in the aligned graph is defined by the number of reads aligned to that position. For example, the local coverage is 5 for all positions in Figure 3.4 as there are 5 reads (excluding seed sequence) aligned to the seed sequence. For a node, if one of the incoming edge's weight is more than half of the local coverage at that position, the node gets a positive score. Otherwise, we will assign a negative score. The path score is the sum of the scores of nodes in the path. The detailed pseudocode can be found in HGAP.

- **Combine HMM score and path score:** users can choose the weights of the path score in

graph G and the HMM score in M. Ideally, α and β should be adjusted based on the local coverage. For high local coverage, we can assign bigger α than β. By default, we use the same weight, which is the chosen parameters for all of the experiments in this work.

- **Running time analysis** The time complexity of our algorithm is $O(\delta|N||M|)$, where $|N|$ is the number of nodes in the G and $|M|$ is the number of states in M . δ is the average number of possible paths that contain a codon ending with a node. Based on our test, for 20X coverage, δ is about 4 to 6 per node. We also found at high local coverage area, the edge with weight 1 can be ignored as the probability to include that edge in the optimal path is extremely low. Thus, we removed those edges to further speed up the algorithm. After this pruning, δ can be as low as near 1 per node on average.

3.4.3 Filtration stage for removing unrelevant protein domain families

During homology search, we align the constructed alignment graphs from PB data with all characterized domains in Pfam. But for any given species or a community, not all domains are relevant. In order to reduce the search space, we apply a filtration stage to remove domains that are not relevant to the given data. In practice, we apply HMMER to all consensus sequences extracted from the alignment graph with a very loose E-value cutoff (1000 is used in all experiments). Only domains that yield at least partial alignments will be used as input to our dynamic programming. Other domains without any hit will be discarded because it is unlikely that they will be true domains in this data set. Each consensus sequence might be aligned to multiple domains, for regions that cannot be aligned to any domain, we also remove them from next stage. Thus, after filtration, the trimmed alignment graph and the corresponding domain will be used as input to our tool. Note that this domain may just incur a very small score and non-significant E-value. It will be re-aligned using our

algorithm and the final alignment score will decide whether it is a true domain. In our experiments, we refer to the input as sequence-domain pair. In all of our experiments, true domains found by corrected sequences using suggested cutoffs is always less than the input sequence-domain pairs.

3.5 Experimental results

Our method is used to improve the sensitivity of profile homology search by correcting insertion or deletion errors in PB sequencing data. Thus, we will focus on evaluating the performance of our implementation in both error correction and homology search. We applied Frame-Pro to three datasets: a simulated *E. coli* PB RS sequencing dataset, a real *M. ruber* PB RS sequencing dataset, and a Human arm PB RSII metagenomic sequencing dataset. The three datasets enable us to evaluate the performance of error correction for data with different sequencing coverage. As both the genomes and their protein domain annotations of the first two data sets are available at NCBI and Pfam, we are able to quantify the accuracy of error correction and profile HMM search. Specifically, we used BLAST to evaluate its error correction performance by aligning corrected sequences to reference sequence. We also quantified the performance of protein domain annotation by applying HMMER3 to corrected sequences and the reference genomes.

We benchmarked our method with the error correction stage DAG-Con in HGAP [24]. The error correction in HGAP and our method do not rely on short sequences generated by another platform. And the error correction in HGAP has been extensively tested and has satisfactory performance for sequencing data with reasonable coverage. DAG-Con only outputs corrected sequences. In order to evaluate the performance of profile homology search, we apply HMMER to the corrected sequences of DAG-Con. Although Frame-Pro outputs both corrected sequences and the profile alignments, we rerun the corrected sequences against input domains using HMMER to

ensure a fair comparison by the same alignment tool.

For our experiments, all detailed commands, parameters and output can be found along with the source code of Frame-Pro. To achieve a fair comparison on data of low coverage, we set the coverage threshold of DAG-Con as 1 to make sure the outputs are comparable.

3.5.1 Simulated *Escherichia coli* sequences

In order to evaluate the accuracy of Frame-Pro in detecting and correcting insertion and deletion errors, we generated a simulated *E. coli* K-12 MG1655 (NCBI tax. ID 511145) sequence dataset by PBSIM [114]. We used the reference genome sequence (NC_000913.3, genome size 4,641,652 base pairs, [60]) generated by Sanger sequencing as input for PBSIM. The quality information and the sequencing length distribution from real PB RS sequencing after secondary analysis [115] were used as the simulation parameters. PBSIM generated 34,898 sequences with 92,810,130 base pairs with average sequencing coverage of 20X. The average length of reads is 2,660 bp and the reads' average error rate is 14.42%.

In the dataset, 3,280 sequences fulfilled seed criterion. To control the scale of experiment, we randomly select 451 seed sequences for graph construction. After graph construction and protein domain filtration, 7,093 sequence-domain pairs were kept as input to our program.

3.5.1.1 Performance of error correction

Both Frame-Pro and DAG-Con produced 7,093 corrected sequences from the input PB simulation data. We first evaluated the performance of error correction of both tools by comparing their corrected sequences to the reference sequences. The comparison is conducted using BLAST [4]. Figure 3.5 summarized the comparison of both tools on each read. For this data set, our tool can correct more errors in about half of the reads while DAG-Con corrects more errors in less than

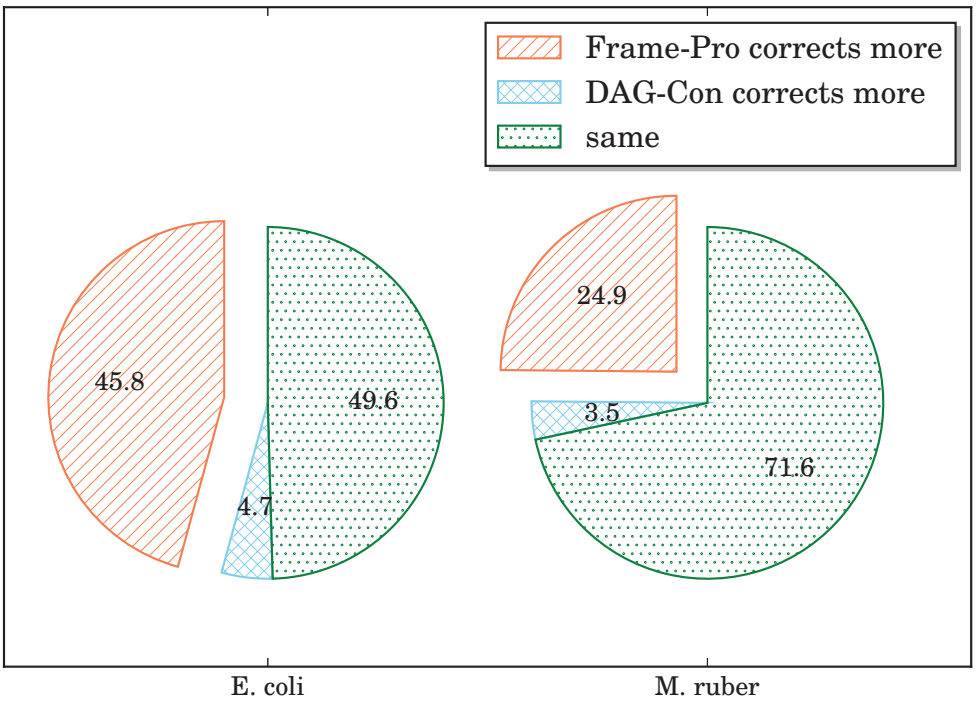

Figure 3.5: Comparison of error correction performance for Frame-Pro and DAG-Con in the simulated *E. coli* dataset and *M. ruber* dataset. Frame-Pro corrects more errors in larger fraction of PB reads.

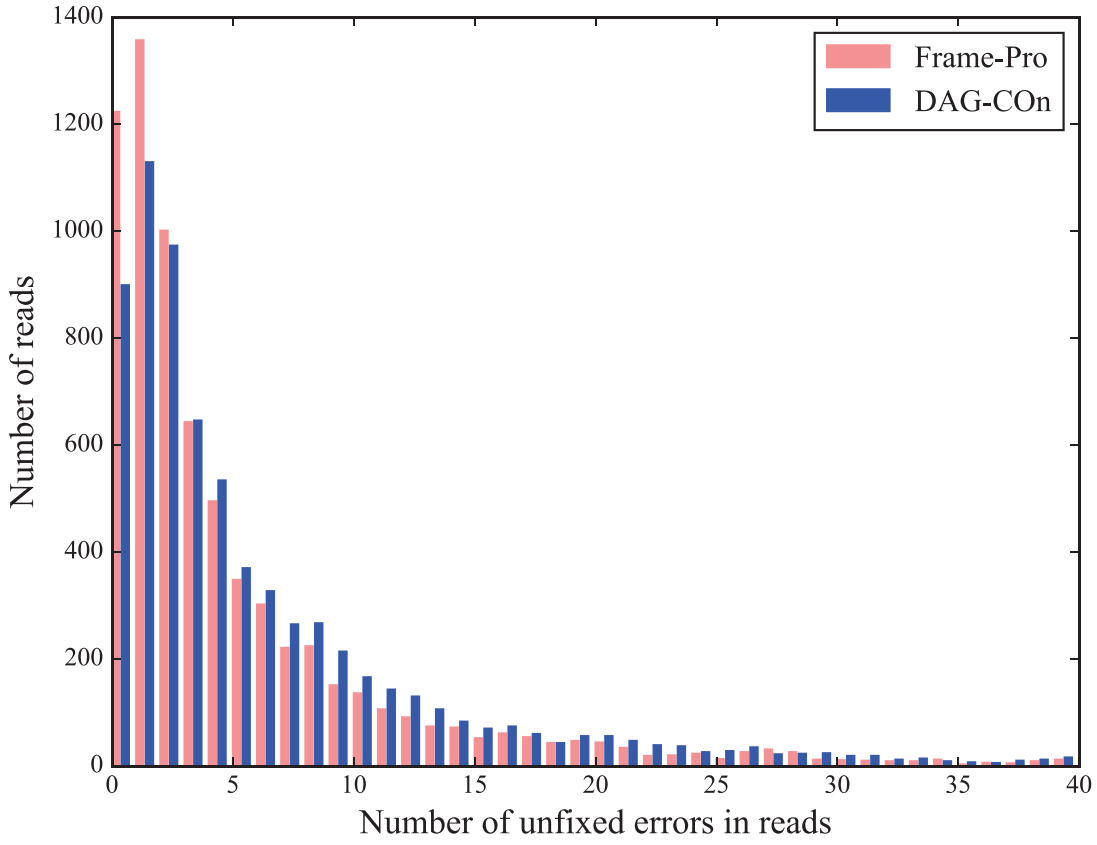

Figure 3.6: Histogram of unfixed errors after error correction in the simulated *E. coli* dataset. X-axis represents the number of remaining errors in each read. Y-axis is the corresponding numbers of reads. Bin width is 1 and the figure only included the first 40 bins (i.e. up to 40 errors) due to space limitation.

5% of reads. In total, our method corrects 7,884 more errors than DAG-Con. If we only count insertion and deletion errors, our method corrects 8,374 more errors than DAG-Con. DAG-Con corrects 490 more mismatches than our method. It is expected because the profile HMM search is much more sensitive to frameshifts caused by gaps as they will significantly impact the alignment length and score. Figure 3.6 compares the number of remaining errors in corrected reads produced by both tools. It is clear that our method can produce more reads with no error or just 1 error.

3.5.1.2 Performance of profile HMM search

One of our major goals is to improve performance of profile homology search. This section will focus on evaluating the performance of protein domain homology search of corrected sequences. After correcting frameshifts, we expect that the homology search program can generate longer alignments with higher scores and smaller E-values for protein domains of interest. So the users can distinguish true domains from random alignments with higher confidence.

HMMER 3.1b2 was used to generate profile HMM alignments from corrected sequences. The domain composition of *Escherichia coli* K12 (NCBI tax. ID 83333) proteome from Pfam (Release 29.0, [45]) was used as the reference. 2,347 protein profile HMM domains were found in 7,093 output sequences. Some sequences have multiple hits. For all the data, including the PB simulation data, and the corrected sequences by both tools, we only keep the best alignment for each domain in our comparison. The changes of alignments' E-values, alignment lengths, and bit scores due to error correction are presented in Figure 3.7. On average, the length of the domain alignment increases from DAG-Con's 108.51 amino acids (a.a) to 150.36 a.a, which is closer to the average alignment length of 163.62 a.a in the reference proteome from Pfam. A two-sample Kolmogorov-Smirnov test (K-S test) on the alignments' lengths and E-values from our method and DAG-Con was applied to examine the statistical significance of difference from two methods.

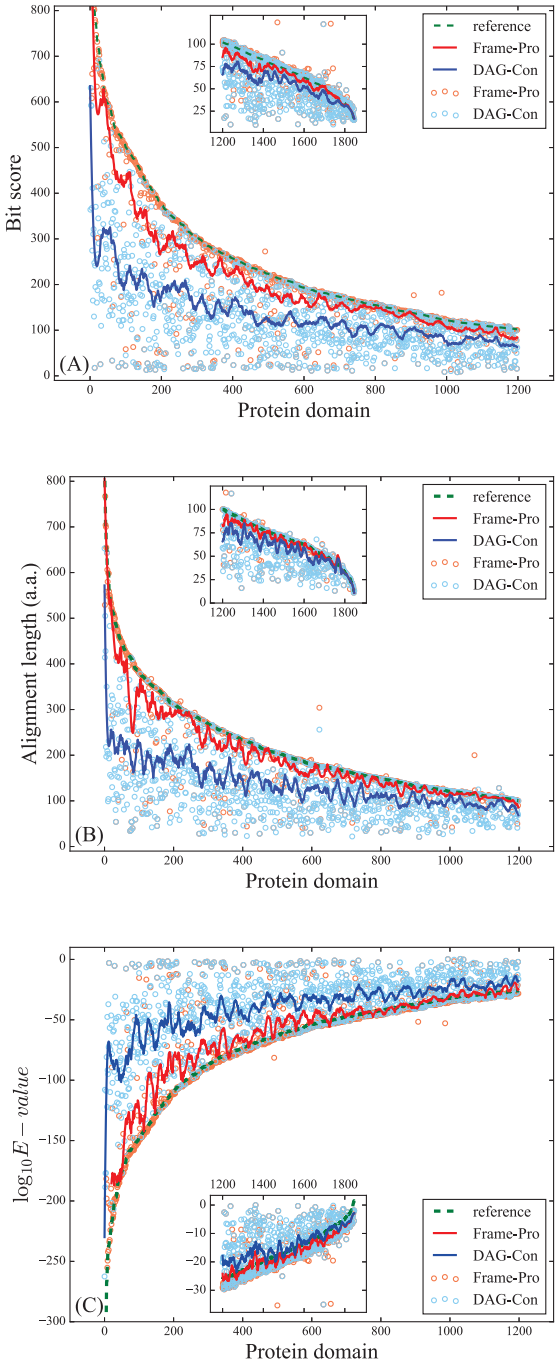

Figure 3.7: The comparison of alignments' bit scores (A), lengths in a.a. (B), and E-values (C) for reference sequences and corrected sequences produced by Frame-Pro and DAG-Con in the *E. coli* simulated dataset. X-axis represents the domains. All domains are sorted by the reference values from Pfam. Red circles represent the values of HMM alignments for corrected sequences output by Frame-Pro. Blue circles represent the values of HMM alignments for corrected sequences output by DAG-Con. As there are many data points, all numbers produced by one tool are processed by SavityzkyGolay filter to generate a smoothed curve for clarity of presentation.

The p-values for the alignments' length, E-value, and bit score distributions were 5.4436×10^{-25}, 1.3237×10^{-25}, and 1.8881×10^{-25}, respectively. Thus, the improvement by applying our method is statistically significant.

3.5.2 *Meiothermus ruber* DSM1279 sequences

The simulated data enables us to thoroughly test the parameters of our tool and also to evaluate various aspects of the performance. For the next two experiments, we apply our tool to real PB data with different coverage. As the real sequencing projects, in particular the transcriptomic sequencing projects and metagenomic sequencing projects, can contain transcripts or genomes with heterogeneous coverage, it is thus important to evaluate tools for data of different coverage.

In this experiment, we tested Frame-Pro on real *Meiothermus ruber* DSM1279 PB Sequencing data. *Meiothermus ruber* (NCBI tax. ID 504728) is usually found in hot springs [141]. It has genome size of 3,098,881 bps. The raw sequencing data from 1 SMRT cell in HGAP [24] was used for this experiment. The raw data in total contains 177.4M bps with 59.6% GC content with approximate coverage of 60X.

Standard SMRT data processing pipeline [117] was used to filter the raw sequencing data to generate filtered subreads, which contain sequences passing the commonly used length and quality criteria. The filtered dataset has 90,114,302 bps and 36,180 sequences with 30X coverage. After protein domain families filtration, 33,911 sequence domain pairs passed the threshold and were input to downstream error correction pipelines.

3.5.2.1 Evaluate error correction performance by aligning outputs against the reference genome

To compare the error correction performance of Frame-Pro and DAG-Con on this real PB sequencing dataset, we used BLAST to search all outputs against the reference genome of *Meiothermus ruber* (NC_021081.1). The comparison of error correction for each read is summarized in Figure 3.5. In total, our method corrects 14,613 more errors than DAG-Con. If we only count insertion and deletion errors, our method corrected 15,924 more errors than DAG-Con. DAG-Con corrected 1,311 more mismatches than our method.

Comparing to the previous simulation experiment, the difference of the error correction performance between Frame-Pro and DAG-Con decreased. The main reason is that the performance of DAG-Con usually improves with increased coverage (20X to 30X).

3.5.2.2 Evaluate the performance of profile homology search

The corrected sequences from Frame-Pro and the consensus sequences from DAG-Con were searched against protein domains in Pfam by HMMER3.1b2. The higher coverage of this dataset did improve the performance of DAG-Con.

For 2,984 domains identified by both tools, we compared the best hit for each of them from two tools. The annotated domains from the reference proteome were downloaded from Pfam and were used as the reference. The changes of alignments' bit scores due to error correction are presented in Figure 3.8. No significant improvement can be observed for bit scores. Similar observations were made for alignments' lengths and E-values. Thus, the other two figures were omitted. Compared with the average alignment length of 148.68 a.a from DAG-Con's consensus sequence, the average length of 157.92 a.a by Frame-Pro is much close to the reference's 158.69

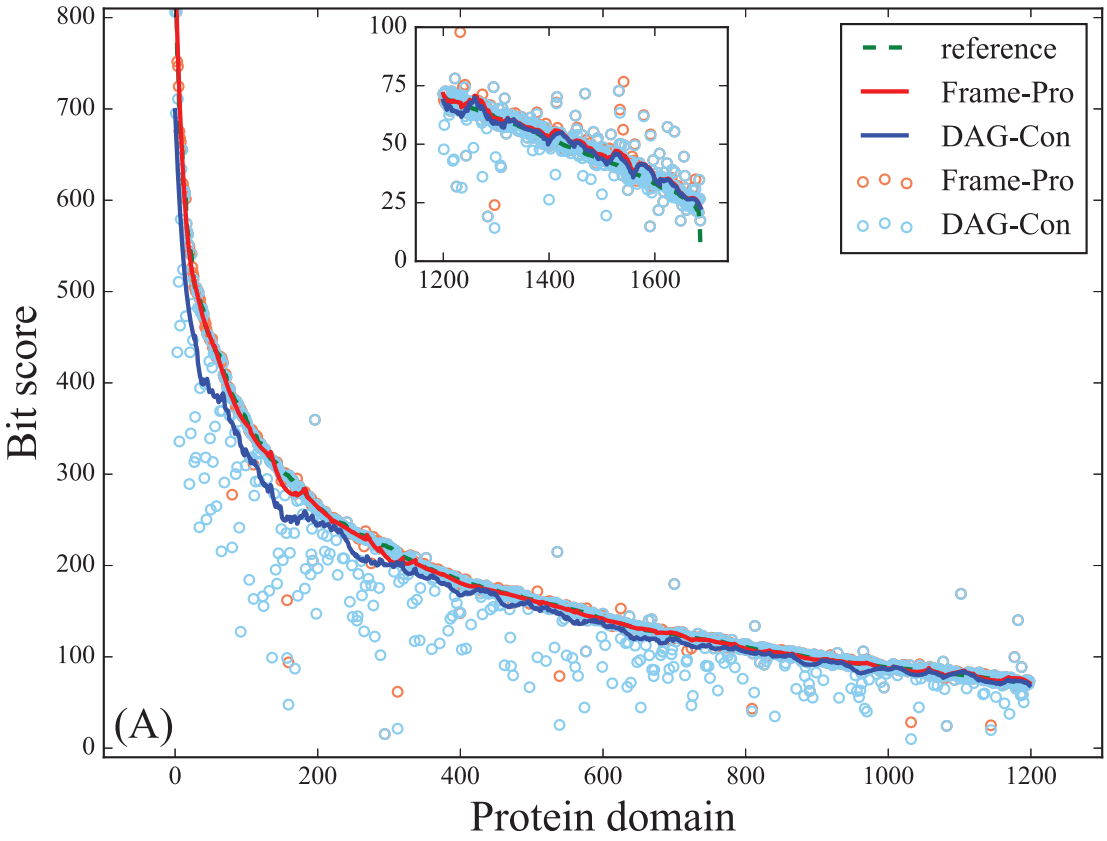

Figure 3.8: The comparison of alignments' bit scores for reference sequences and corrected sequences produced by Frame-Pro and DAG-Con in the *M. ruber* dataset. X-axis represents the domains. All domains are sorted by the reference bit scores from Pfam. Red circles represent the values of HMM alignments for corrected sequences output by Frame-Pro. Blue circles represent the values of HMM alignments for corrected sequences output by DAG-Con. As there are many data points, all numbers produced by one tool are processed by SavityzkyGolay filter to generate a smoothed curve for clarity of presentation.

a.a. We conducted a two-sample K-S test on the alignments' lengths, E-values, and bit scores from our method and DAG-Con. The p-values for alignments' lengths, E-values, and bit scores' distributions were 0.2859, 0.2321, and 0.2007, respectively. Although Frame-Pro still generated longer alignments with better scores, DAG-Con achieved satisfactory performance of homology search for this data set because of the sufficient coverage.

3.5.3 Human arm metagenomic dataset

When PB is applied to metagenomic sequencing, one challenge is that some datasets do not have enough coverage for effective downstream analysis. Here, we applied Frame-Pro to analyze the protein domain composition in the human skin metagenomic data, which were sequenced from the human arm and foot [144].

The human arm sample was sequenced by linear PB RSII TdT (terminal deoxynucleotidyl transferase) sequencing platform. Sequences can be mapped with CHM1 human genome were removed as host human-derived DNA. This dataset cannot be further assembled by the HGAP pipeline due to the insufficient coverage, providing challenges to downstream analysis including protein domain classification. Thus in this experiment, we focus on the arm data set, which includes 16,388 sequences with 2,662,7191 bps.

3.5.3.1 Generate SAG and filtrate profile HMM domain

Compared with previous two datasets, the average read length of the human arm metagenomics dataset is only 1,629 bps. To generate sufficient seed reads, the cut-off was changed to 3,000 bps. Although Frame-Pro is not as sensitive as DAG-Con to the sequences' coverage, the filtration steps were affected as we use HMMER to search the consensus against Pfam with E-value 1000. After filtration, 602 sequence domain pairs were kept for further analysis. For each sequence domain

pair, a corrected sequence using Frame-Pro and a consensus sequence using DAG-Con, both from the same graph, were generated.

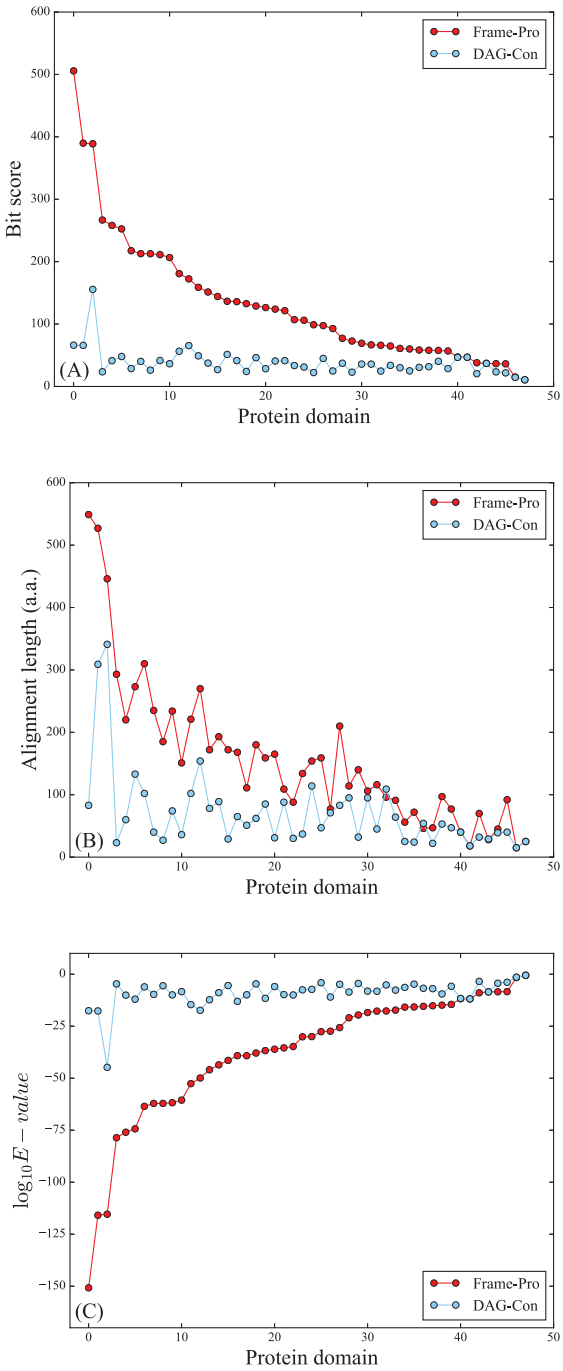

Figure 3.9: The comparison of alignments' bit scores (A), lengths in a.a. (B), and E-values (C) for 48 domains commonly identified by Frame-Pro and DAG-Con in the arm data set. X-axis represents the domains.

3.5.3.2 Protein domain search using GA cutoff

Unlike previous data sets, we don't know all the composite species in the arm sample. Thus we cannot obtain all the ground truth protein domains in this data set, making the comparison with the reference domains difficult. In addition, without complete reference genomes of all species, we don't have the actual sequences corresponding to these reads and thus we cannot evaluate the number of unfixed errors. In order to examine the results, we thus use a stringent threshold to examine the domain sets identified for corrected sequences output by Frame-pro and DAG-Con. We use the gathering (GA) cutoffs from Pfam as the threshold for domain composition analysis because GA cutoffs are family specific bit score thresholds aiming to minimize false positive rate and to maximize the coverage [44]. In Frame-Pro's results, 84 domains are above the GA cutoff, comparing to 49 domains in DAG-Con's results. Frame-Pro only missed one domain in DAG-Con's result with score slightly less than the corresponding GA cutoff. According to Pfam, all domains uniquely found by Frame-Pro were annotated in microbial species. The list of domains can be found in our website.

For the commonly identified 48 domains, we compared the alignments' bit scores, alignment length, and E-values on corrected sequences output by two tools (See Figure 3.9). We conducted a two-sample K-S test on the three metrics for all alignments. The quality of alignment improve significantly. The p-values for the the alignments' length, E-value, and bit score distributions were 6.05×10^{-6}, 3.66×10^{-12}, and 7.64×10^{-13}, respectively.

Without reference sequences, there could be one possibility that Frame-Pro over-corrects the errors in order to maximize the alignment score. In order to test this possibility, we examined the identified domains for one reference species: *Corynebacterium aurimucosum*. Although the complete composite species of the arm sample is unknown, the phylogenetic analysis in [144]

and other articles [47, 142] showed that *Corynebacterium aurimucosum* is relatively abundant in this sample. In addition, this species has annotated domains (NCBI tax. ID 548476) in the Pfam database. Thus, we focus on evaluating how many of the annotated domains in this species can be identified by tested methods. By using GA cutoff, Frame-Pro can recover 41 domains while DAG-Con can recover 24 domains. The set identified by DAG-Con is a subset of ours. Thus, this experiment shows that extra domains identified by us are not likely false predictions. This experiment adds evidence that Frame-Pro can identify more domains for data with very low coverage and thus provides complementary analysis for metagenomic data.

3.6 Conclusion and discussion

In this work, we developed a profile homology search tool for PB sequencing data. By correcting insertion or deletion errors, our implementation can improve homology search performance, including alignment scores, lengths, and E-values. In particular, for sequencing data with low sequencing coverage (around or lower than 20X), our tool can significantly correct more errors and improve the sensitivity of homology search by finding more correct domains. Being able to conduct sensitive homology search for sequencing data of low coverage is important for various sequencing projects including metagenomic and transcriptomic sequencing. Usually, as the transcripts or species have highly different abundance and thus heterogeneous coverage, conducting homology search needs to consider the input of a full spectrum of coverage. Many rare species in an environmental community or rare transcripts are particularly interesting for biological discovery.

Although our current implementation is based on profile homology search, which compares sequences with profile HMMs. Our method can be easily extended to pairwise alignment as well.

In that case, the profile HMM will be replaced a single sequence. Existing pairwise alignment algorithm can be extended to align an alignment graph with a single sequence.

Like the error correction stage in HGAP, our tool does not rely on hybrid sequencing either, making it convenient for various applications. However, one limitation is that our tool is not a general error correction tool because it can only correct errors in regions that are homologous to reference sequences. This is not a problem for species with highly packed coding regions. But for genomes with large fractions of noncoding regions, our tool is not designed for error correction in the whole genome.

Finally, we only tested our application in PB data. But our method can be extended to other sequencing data with similar types of errors. For example, we will test our tool on the data produced by Nanopore technology.

Chapter 4

Deep Learning Based Approach For Protein Domain Prediction

4.1 Introduction

Third-generation sequencing technologies, such as Pacific Biosciences single-molecule real-time sequencing (PacBio) and Oxford Nanopore sequencing (Nanopore), produce longer reads than the traditional sequencing technologies. The increased length enables closing gaps in genome assembly [67, 149], detecting epigenetic modifications [151], and quantifying gene isoforms with higher accuracy in the transcriptomic sequencing [107, 140]. It also shows great potentials in sequencing the microbial communities [23, 144].

One of the most significant challenges to utilize long reads from third-generation sequencing is the high sequencing error rate [7, 67]. Most errors are insertions and deletions, which can cause frameshifts during translations. Without knowing the errors and their positions, the frameshifts can lead to only short or non-significant alignments in the downstream analysis [154]. As a result, traditional downstream analysis pipeline on the raw PacBio or Nanopore reads leads to incorrect results [34].

Characterizing the protein domain in sequences is one of typical downstream analysis with a continuing concern as proteins play pivotal roles in many biological processes. Homology search

is one of the standard methods to annotate the functions of protein sequences by comparing query sequences against reference sequences [4] or homologous sequence families [35]. Algorithms of machine learning is an alternative approach without alignment to predict the protein domain in sequences [30, 147]. The major challenge of these methods is how to convert the sequence to features that can capture the information of protein families. Recent developments in the algorithm of deep learning led to another approach with automatic feature extraction from the sequence [91, 133].

4.1.1 Deep learning for sequential data

Different deep learning model have their own advantages to resolve specific types of problems for sequential data. So we expect that those algorithms can also help on different problems in bioinformatics. Here, we will describe two major deep learning structures, convolutional neural networks (CNNs), recurrent neural networks (RNNs), that already been proved achieve state of the art performance on sequential data.

4.1.1.1 Convolutional Neural Networks

One of the most successful model in recent years is Convolutional Neural Networks (CNNs). The major difference between CNNs and traditional neural networks is to replace the general matrix operation in the tradition layer with the mathematical operation called convolution.

4.1.1.1.1 Convolution operation In mathematics, the convolution of function x and w, $s(t)$, is defined as following equation:

$$s(t) = \int x(a)w(t-a)dt = (x*w)(t)$$

In convolutional network terminology, the first argument (in this example, the function x) to the convolution is often referred to as the input and the second argument (in this example, the function w) as the kernel. The output is sometimes referred to as the *feature map*.

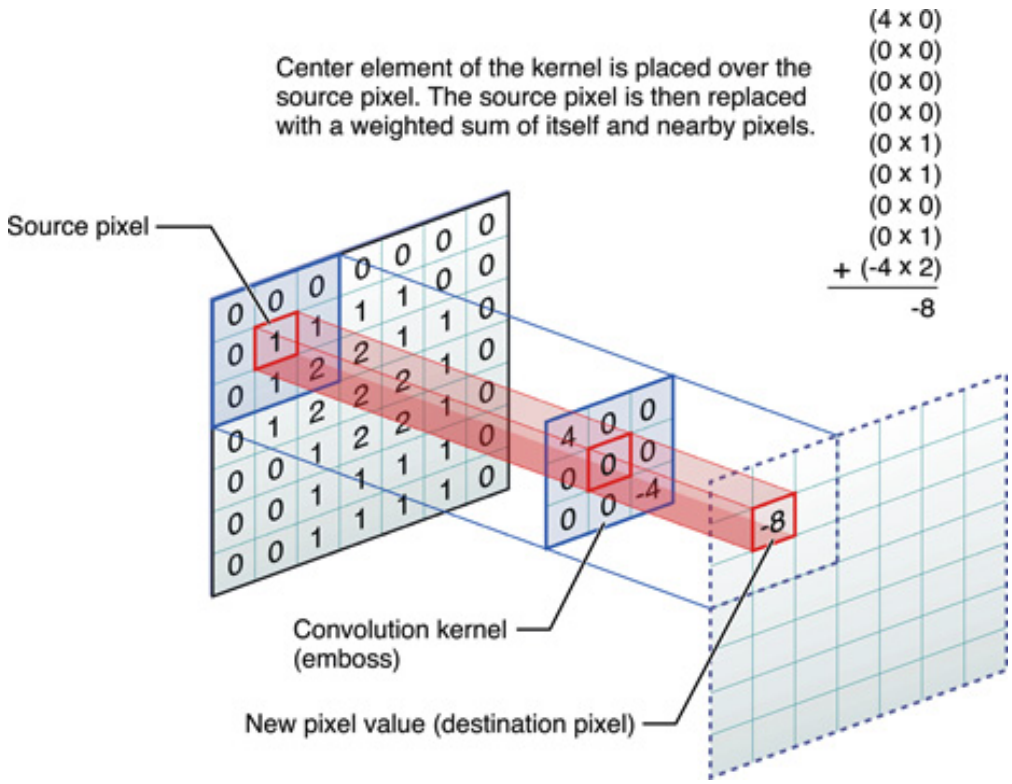

Figure 4.1: How a kernel operates on one pixel. Reprinted from [6].

The convolution operation in the matrix can treat as a kind of matrix multiplication. As the Figure 4.1 showed, the kernel convolution take source pixel from input matrix and simply multiplied the kernel matrix to generate the element for the convolutional layer output.

4.1.1.1.2 Major features of CNNs There are three major features that make CNNs so successful: sparse interactions, parameter sharing, and equivariant representations.

Sparse interactions means that the neuron in the convolutional layers only needs to interact with several neurons in the input and output layers. In the contrast, a neuron in the traditional neural

network needs to interact every neuron in the input and output layers. In this way, the CNNs can easily learn some small and meaningful features which are only from tiny part of the input matrix. Also, it can save a lot of computation time.

Parameter sharing refers to using the same parameter for more than one function in a neural network. At every position of the input in CNNs, it will use the each element of the kernel matrix. By doing this, the layer can learn something with a whole picture.

Equivariant representations describes the phenomenon that if the output will change corresponding to the change of input.

4.1.1.1.3 CNN for sequence classification For a character in the sequence, we can use a k-dimension vector $\boldsymbol{x}$ to represent it. So a sequence of length n is represented as:

$$\mathbf{x}_{1:n} = \boldsymbol{x}_1 \oplus \boldsymbol{x}_2 \oplus ... \oplus \boldsymbol{x}_n$$

Here operator $\oplus$ is the concatenation operator. We applied convolution filter w on a window of size h, to transform the input $\mathbf{x}_{1:n}$ to a new feature c_i:

$$c_i = f(\mathbf{w} \cdot \mathbf{x}_{i:i+h} + b)$$

Here b is the bias term and f is the activation function usually using non-linear function like sigmoid or hyperbolic tangent. By repeated apply the operation, we can finally generate feature map. Then the max-over-time pooling was applied to keep the most important features.

TextCNN [74] is one of the earliest models adopting the structure we discussed above for sentence classification. The key idea of TextCNN is using kernels with different sizes to capture

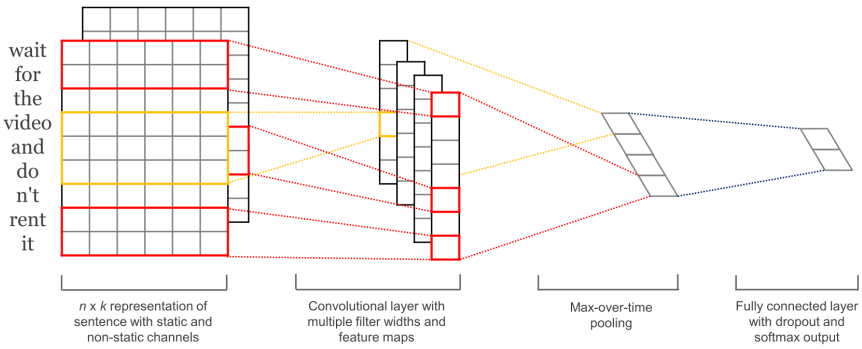

Figure 4.2: Structure of TextCNN model. Reprinted from [74].

pattern in different ranges. After global max pooling, the features then used as input for a fully-connected layer and then convert to the probability of labels. The variation of TextCNN model has been adopted for solving different bioinformatics problem, such as protein domain classification [133], virus sequence identification [127], and DNA-protein binding prediction [153]

4.1.1.2 Recurrent Neural Networks

Recurrent Neural Networks, or RNNs [129] are the family of the deep learning structures to process sequential data. Parameter sharing across the different parts of the model is the key idea that makes RNNs to be able to deal with the sequential data. Specifically, the RNNs we talked about is operating on a sequence that contains vectors $\boldsymbol{x}^{(t)}$ with the time step index t ranging from 1 to τ.

4.1.1.2.1 Basic structure of RNNs A simple recurrent neural network will just process the information from the input $\boldsymbol{x}$ and incorporate it into the state $\boldsymbol{h}$ of hidden unit and passed to the

next unit. The hidden unit state $\boldsymbol{h}$ at time step t, $\boldsymbol{h}^{(t)}$,

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$

RNN also usually have extra architectures like output layers that output the result of $\boldsymbol{h}$ for further processing or predictions.

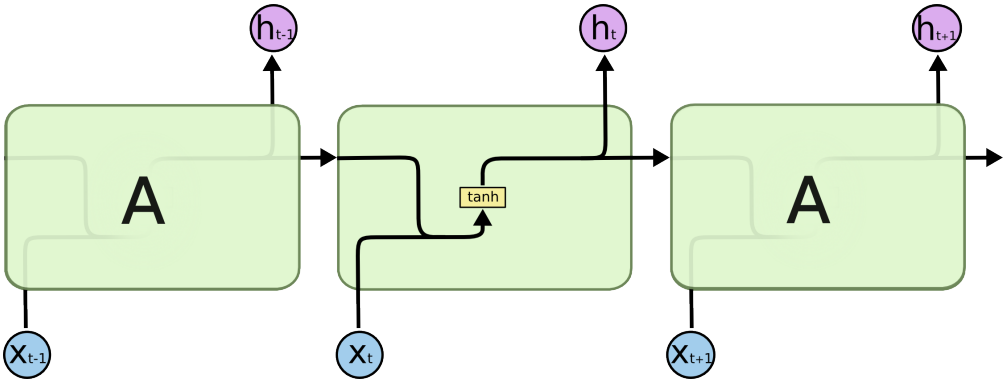

Figure 4.3: The repeating module in a in a simple recurrent neural network structure. Reprinted from [113].

We can use a function $g^{(t)}$ to represent the unfolded recurrence after t steps, given us another representation of $\boldsymbol{h}^{(t)}$,

$$\boldsymbol{h}^{(t)} = g^{(t)}(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, ...\boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$$

In this form, $g(t)$ takes the input of whole past sequences $(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, ...\boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}))$ as input and output the current state. However, we can unfolding the structure and repeated apply function f to achieve same output. By doing this we can use the same transition function f with the same parameters at every step.

However, a simple RNNs cannot learn long time dependency as in the optimization. As we described above, at each time step we repeatedly apply the same operation f. We can use a multiplication on matrix $\mathbf{W}$ to represent such operation. Suppose that matrix $\mathbf{W}$ has an eigendecomposition

$\mathbf{W} = \mathbf{V}\text{diag}(\boldsymbol{\lambda})\mathbf{V}^{-1}$. After t steps, the input already multiply $\mathbf{W}^t$, so:

$$\mathbf{W}^t = \mathbf{V}\text{diag}(\boldsymbol{\lambda})^t\mathbf{V}^{-1}$$

Means that given enough large steps t, any eigenvalue λ that not close to 1 will either vanish (less than 1) or explode (larger than 1). The gradient in such graph also scaled according to $\text{diag}(\boldsymbol{\lambda})^t$ [52]. In this case, it difficult to optimize such gradient. To solve this challenge, gated RNNs is proposed and becomes one of the most effective practical models that used for sequential data.

4.1.1.2.2 LSTM The long short-term memory (LSTM) architecture [66] is one branch of such gated RNNs that is extremely successful in the application like speech recognition, machine translation, and handwriting generation. The key idea of LSTM is to introduce a self loop so that gradient can flow for long duration.

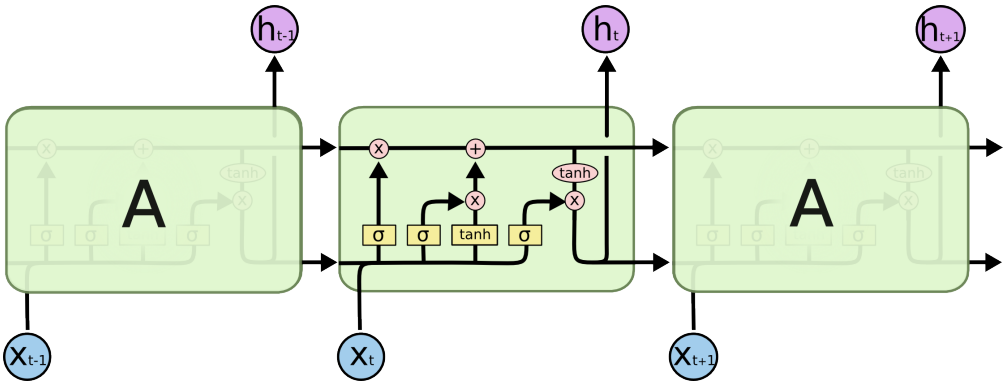

Figure 4.4: The repeating module in a LSTM cell. Reprinted from [113].

The self loop (internal recurrence) is located in "LSTM cells" with outer recurrence like ordinary recurrent network. The cell is controlled by a combination of input gate and memory control gates, formulated by equations below:

$$i_t = \sigma(U_i h_{t-1} + W_i x_t + b_i)$$

$$f_t = \sigma(U_f h_{t-1} + W_f x_t + b_f)$$

$$o_t = \sigma(U_o h_{t-1} W_o x_t + b_o)$$

$$\tilde{c}_t = \sigma(W_c x_t + U_c h_{t-1} + b_c)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$h_t = o_t \odot \sigma(c_t)$$

where i_t, f_t and o_t are three gates that control input, forget and output respectively. Each of the gate output signal depends on its state transition matrix U, input weighting matrix W and bias b. The final cell state c is updated by element-wise multiplication, denoted by operator "$\odot$". σ is the activation function, which can be chosen from sigmoid, ReLU, tanh and so on.

As we can see, LSTM adds two more gates to control the output and previous inputs, thus has the ability to capture the long term dependencies in the sequence. So far, LSTMs are the most successful variation of RNNs.

4.1.1.2.3 GRU The gated recurrent unit (GRU) architecture [25] is an alternative and the main competitor to the LSTM. The performance of GRU is comparable to the LSTM on many sequential datasets, with simple unit structures and less computation requirement.

The main difference of GRU with the LSTM is that a single update gate replace the role of the forget and input gates. So the equations are updated to following:

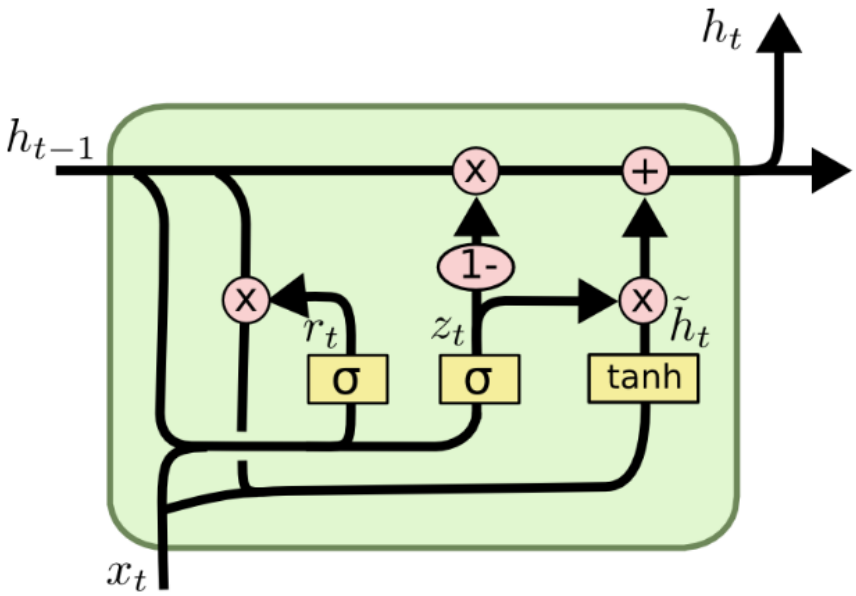

Figure 4.5: The repeating module in a GRU cell. Reprinted from [113]..

$$z_t = \sigma(U_z h_t + W_z x_t + b_z)$$

$$r_t = \sigma(U_r h_t + W_r x_t + b_r)$$

$$\tilde{h}_t = \sigma(W_h xt - 1 + U_h(r_t \odot h_{t-1}))$$

$$h_t = z_{t-1} \odot h_{t-1} + (1 - z_{t-1}) \odot \tilde{h}_t$$

4.1.1.3 Protein function prediction using deep learning

Recently, deep learning techniques have been shown to achieve state-of-the-art results in many machine learning problems without the need for complex feature engineering [52, 85]. Deep learning based method also have demonstrated their ability on many bioinformatics problems, such as DNA-protein and RNA-protein binding sites prediction [3, 153], taxonomic classification [19, 43], and SNP and small-indel variant detection [122].

The success of deep learning techniques inspired researchers to apply the approach to modeling the protein sequences. Long short-term memory (LSTM) was first introduced by [65] to model the protein homology detection. The model can automatically extract the long-term and short-term dependency of the sequences and predict whether the sequences belong to the modeled protein family. Recently, a bidirectional LSTM model, ProDec-BLSTM [91], was proposed to improve the prediction capability of the model further. In the model, every hidden value of the LSTM was used to capture the long and short dependency information of the proteins. The experiment result shows ProDec-BLSTM outperforms various existing methods, including HMMER.

DeepFam [133] adopted convolutional neural networks (CNN) to model families of protein sequences. Unlike LSTM models we mentioned earlier that can only model single protein family, DeepFam used a single CNN model to predict multiple protein families. Various sizes of convolutional filters were used in DeepFam to extract conserved regions to model protein families. It outperforms HMMER and previous alignment-free method. Also, the execution time of DeepFam is fast as it not affected much by the number of families while HMMER needs longer time as the number of families increased. For example, DeepFan is more than ten times faster compared with HMMER when 1,000 query sequences searching against thousands of protein families [133].

However, existing methods like DeepFam cannot handle errors and incomplete sequences. All the deep learning method we mentioned were only tested using complete and correct protein sequences. Moreover, DeepFam used a close dataset to test their performance, which assumes that test dataset are held-out samples from the same distributions of the training dataset. In a real application, a sequence may be sampled from non-coding sequence or other protein families that not be modeled in the neural network. We call this task detection task hereafter, as the main goal is to detect the targeted protein-encoding sequences from other irrelevant sequences. In theory, irrelevant input is supposed to be rejected. But current deep learning classification models employ softmax

function to assign the label and are not able to decline such cases, yielding high false-positive rate [10, 63].

4.1.2 Related work

4.1.2.1 Profile homology search

To predict the protein domain of a given sequence, we can directly compare the sequence with protein-encoding sequences using pairwise alignment tool like BLAST [4]. However, in many applications, we usually do not care if the query sequence close to one specific encoding sequence of the target protein. Instead, we want to align the query sequence to a family of protein sequences. A profile hidden Markov model (pHMM) can be built from multiple sequence alignments to model a protein family. By explicitly define the match, insertion, and deletion state, pHMM can handle the comparison of the query sequence to the profile of protein families accurately, even when the query is remote homology of the target protein families. HMMER is one of the most widely used profile search tools based on pHMM [36]. Profile of the protein families can be downloaded from many protein databases, such as Pfam [40], and TIGRFAMS [56].

As an alignment-based method, the speed of the profile homology search suffers when the number of families increases. Extensive research has studied to improve the efficiency of the profile homology search. The newest version (3.1b2) of HMMER [37] is more sensitive than BLAST and is about 10% faster. However, recent research [133] suggests it is still slower than the alignment-free method.

4.1.2.2 Homology search with error correction

Due to the high error rates, the homology search result on raw PacBio or Nanopore reads usually is not desired with wrong annotation and short alignments. In this situation, error correction on raw reads can effectively improve the homology search result.

There are two major correction methods for third-generation sequencing data: hybrid error corrections and non-hybrid error corrections. The hybrid method correct long noisy read by combining short, accurate second-generation sequencing reads with third-generation reads to remove errors [78, 136]. Unlike hybrid sequencing, non-hybrid error correction, or self-correction, only rely on the long reads themselves for error correction [24,48,93]. As the sequencing errors in third-generation reads occur randomly, the inferred consensus sequence from the alignment between the longest backbone reads and shorter reads from third-generation sequencing dataset represent the high-quality sequence. However, its performance is profoundly affected by the coverage of the aligned sequences against the backbone sequences [34].

4.1.3 Overview of our work

As we discussed in the previous section, the current protein predict algorithms cannot handle the high error rate of the third-generation reads. Thus, we designed and implemented DeepFrame, a deep learning based method to predict the protein domain of third-generation sequences.

We optimized the neural networks architecture of DeepFrame based on rigorous validation performed on the classification of a simulated PacBio sequence dataset. The classification accuracy of DeepFrame consistently outperforms the state-of-the-art method like HMMER across various error rates. We then extended the framework to the detection task with Outlier Exposure and tested on real third-generation sequencing dataset. DeepFrame achieves higher F1 score and higher recall

with comparable precision when compared with HMMER.

Compared to previous works, DeepFrame has several merits. First, we designed a three-channel encoding for the task of function prediction using DNA reads. Compared to other encoding methods (discussed in Experiments and results), model with three-channel encoding achieved better performance. Second, evaluated by simulated PacBio data and real PacBio and Nanopore data, DeepFrame can successfully process long erroneous reads without any prepossessing. Although we only used simulated PacBio reads that are easy to generate to train our model, the performance on real Pacbio and Nanopore data is still robust and consistent. Third, unlike previous deep learning work that only designed for classification, DeepFrame can also be used for detection tasks. With the threshold on softmax prediction probability [63] and Outlier Exposure method [64], DeepFrame can distinguish the DNA sequences with targeted protein domain from other random coding or non-coding DNA sequences.

4.2 Methods

DeepFrame is designed to predict protein domains for third-generation sequencing data. It incorporates 3-frame encoding to convert DNA reads into 3-channel tensor as the input of neural networks. From the input, DeepFrame automatically extracts features by using convolution layer and max-over-time pooling layer. A classifier with two fully connected layer was used to generate the probabilities of the sequence against all possible protein domains. To exclude the unrelated coding or non-coding DNA sequences, we then compare the max value of the output probabilities with a threshold. We will discuss the details of our model in the following section.

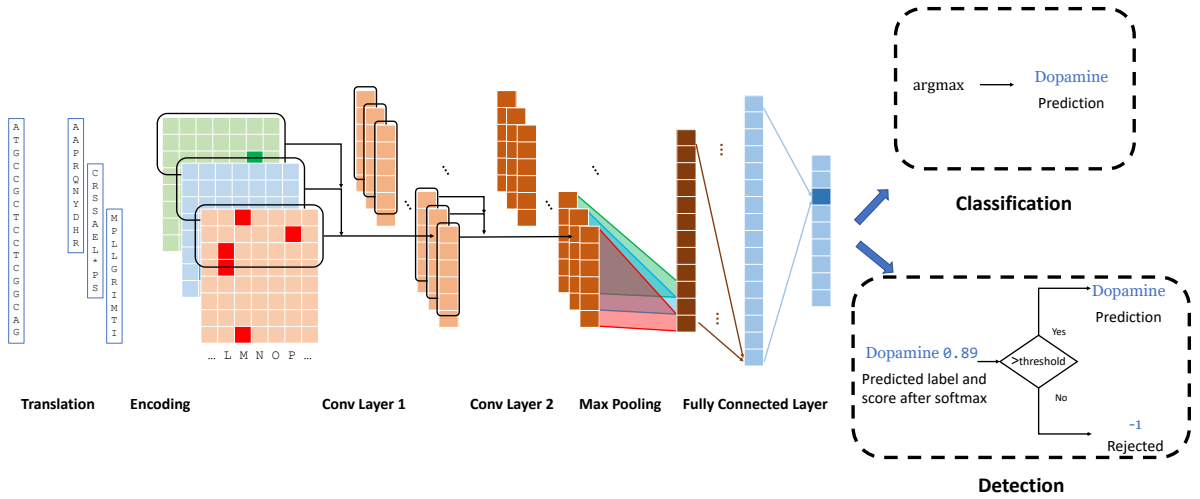

Figure 4.6: The overview of DeepFrame model. The input sequence was translated and encoded to a 3-channel tensor. Two convolutional layer and max pooling layer extract sequence features. These features were used by a fully connected layer with softmax function to infer the probability of each protein families. In the classification task, the model directly outputs the families with the largest score as the prediction. In the detection task, the maximum of softmax score needs to compare with the threshold to determine whether the input contains a trained protein family or should be rejected.

4.2.1 Encoding

To process the DNA reads, we need to encode sequences to numerical values. We designed a 3-frame encoding scheme for the input sequences. Each DNA sequences is 3-frame translated to 3 protein sequences. All the residues in the protein sequence are one-hot encoded using a 21-dimensional vector following IUPAC amino acid code notation. Then we concatenate three matrices into a single 3-channel tensor like an RGB image. The rationale behind this is that we hope the convolution filters can automatically distinguish the right fragment on each frame after translation and extract the corresponding features. Also, the relative order of the residues at the same position of the three frames incorporate the information of the original DNA sequence. Given the sequence length L, then the encoded input is a tensor with size $3 \times L \times 21$. The pseudo-code of 3-frame encoding can be found in Algorithm 4.

Algorithm 4 3-frame encoding

Input: DNA sequence x with length L, peptide to index dict idx, peptide alphabet size $|\Sigma|$, output sequence length n.
Output: Input tensor for neural networks with size $3 \times n \times 21$.

Initialize an array arr with all 0 values and dimensions [3, n, $|\Sigma|$]
for $i = 1$ to 3 **do**
    $x_i \leftarrow x[i:]$
    $y_i \leftarrow$ translation of x_i ▷ translate nucleotides in x_i to amino acids in y_i
    for residue a at position k in y_i **do**
        if $k \leq n$ **then**
            $arr[i,k,idx[a]] \leftarrow 1$ ▷ one-hot encoding for each frame
        end if
    end for
end for
arr is input tensor for neural networks

We also tested other encoding methods: (1) DNA one-hot encoding, which directly trasnfer DNA sequence to one hot vectors of size $L \times 4$. For fair comparison, we used filter sizes that are 3 times as long as we used for 3-frame encoding; (2) 3-branch model, we constructed a network ar-

chitecture with three separate branches processing each of the 3-frame translated protein sequence respectively. Each of the branches consists of identical layers, and all the parameters are shared between the same layer of 3 branches. In the 3-branch model, each of the branches models the translated protein sequences separately before the merging layer right before the two-layer classifier. In contrast, in 3-frame encoding, all three translated protein sequences were processed and combined by the 3-channel convolutional filters in the first convolutional layers. Our experimental results show that 3-frame encoding is a better encoding scheme as it can effectively encode the original DNA sequence information and also helps convolution filters to extract useful features for prediction of the protein domain (Section 4.3.1.1).

4.2.2 Convolutional Neural Networks

DeepFrame consists of two convolutional layers, one max-over-time pooling layer, one hidden linear layer, and one linear output layer with softmax function.

The convolutional layer [46, 74, 86] is the most important building block in the model. For a residue in the single frame protein sequence, we can use a k-dimension (k = $|\Sigma|$ in our case) vector $\boldsymbol{y}$ to represent it. So a sequence of length n is represented as a one-hot encoding matrix. We applied convolution filter $\mathbf{w} \in \mathbb{R}^{hk}$ on a size h window of the matrix, to transform the input $\mathbf{y}_{i:i+h-1}$ to c_i:

$$c_i = f(\mathbf{w} \cdot \mathbf{y}_{i:i+h-1} + b) \tag{4.1}$$

Here b is the bias term and f is the activation function usually using non-linear function like sigmoid or hyperbolic tangent. In our model, we use ReLU [104] as activation function of convolutional layer:

$$f(x) = \sigma(x) = \max(0, x) \tag{4.2}$$

We can extend Eq. 4.1 to multiple channel input (in our case, channel = 3:

$$c_i = f(\sum_{j=1}^{3} \mathbf{w}_j \cdot \mathbf{y}_{i:i+h-1,j} + b) \tag{4.3}$$

For both Eq. 4.1 and 4.3, we applied convolutional filters repeatedly to each possible window of the one-hot matrix $\mathbf{y}_{1:h}, \mathbf{y}_{2:1+h}, ..., \mathbf{y}_{n-h+1:n}$ to produce a feature map, which is the vector of extracted feature values:

$$\mathbf{c} = [c_1, c_2, ..., c_{n-h+1}] \tag{4.4}$$

Then the max-over-time pooling is applied on the feature map to capture the maximum value $\hat{c} = \max \mathbf{c}$ as the feature from this particular filter. The rationale behind this operation is to extract the peak signal as the most important feature for each feature map. The max-over-time pooling is flexible with different input length.

We described how a single filter in the convolutional layer works. In our application, we used multiple filters with varying filter size (or window size) to extract various features with different ranges.

DeepFrame has two convolutional layers. The first convolutional layer uses consistent convolution filter size to extract low-level short distance patterns directly from 3-frame encoding input. Then second convolutional layer extracts high-level, intricate patterns with varying distance from the output of the first convolution layer. By repeatedly applying the operations, we can finally generate a feature map. Then the max-over-time pooling was applied to keep the most important features. Dropout [138] is also used after pooling to prevent overfitting and to learn robust features. A two-layer classifier with softmax function transfers the features to a vector of probabilities over each label. For classification, we select the label with the maximum probability as the prediction from DeepFrame.

4.2.3 Out-of-distribution examples detection

We show that DeepFrame classifier with softmax function can predict the protein labels of given DNA reads. However, the classifier will always assign a label for the input sample, even if the input is not related to any label in the model (we call such inputs out-of-distribution examples, compared to in-distribution examples). For example, in RNA-seq data, not every read encodes targeted protein families in the model. When processing such unrelated inputs, the model will still assign the input to one of the protein families in the model as there is no such label for the out-of-distribution input. In a real application, this behavior will lead to an undesired high false-positive rate. To address the problem, we adopt Outlier Exposure (OE) [64] with a threshold on softmax prediction probabilities [63] to distinguish the out-of-distribution examples.

4.2.3.1 The threshold baseline

Usually, the examples from out-of-distribution dataset tend to have small softmax value because its probabilities vector tends to be more uniformly distributed than the examples from in-distribution dataset. Thus, by specifying a threshold on maximum probability in the softmax's output, it is possible to distinguish out-of-distribution examples form in-distribution examples . Following [63], we extract the maximum value of the softmax probability from the output of DeepFrame for each input exmaple. We separate the in-distribution examples from the out-of-distribution examples by specifying a threshold of the maximum softmax probability. To determine the threshold, another simulated out-of-distribution dataset is created and combined with the simulated holdout test dataset. For each example, we compute the maximum softmax probability. We obtain F1 score from the dataset:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{recall} + \text{precision}} \tag{4.5}$$

The F1 score summarizes the performance of the binary classification of in-distribution examples and out-of-distribution examples for a given threshold. We compute the F1 score across different thresholds and select the threshold of best F1 score as the threshold used in our model.

4.2.3.2 Outlier Exposure

To further improve the performance of out-of-distribution examples detection, we adopt the Outlier Exposure (OE) method introduced by [64]. As we discussed previously, we expect the out-of-distribution examples to have uniform softmax output from the model. However, as such inputs never be processed in the training, sometimes the model will yield unexpected high confidence prediction for out-of-distribution input (Figure 4.7). To address the problem, we expose the model to out-of-distribution examples in the training process to let the model effectively learn the heuristics of the out-of-distribution inputs.

Given a model g and the learning objective $\mathscr{L}$, the objective of OE is to minimize the original loss function with an auxiliary loss terms to regularize the out-of-distribution examples [64]. In original classification task, we use cross-entropy loss function as $\mathscr{L}$. So we set the auxiliary loss $\mathscr{L}_{\text{OE}}$ to the uniform distribution:

$$\mathscr{L}_{\text{OE}} = -\log\left(\frac{\exp \hat{y'}}{\sum_j \exp y'_j}\right) = -\overline{y'} + \log\sum_j \exp y'_j \tag{4.6}$$

Here, $\overline{y'}$ is the mean value of the output logits (the output of CNNs model before the softmax function). And we use $\lambda = 0.5$ in our experiment.

Figure 4.7 presents the distribution of the softmax score of DeepFrame model with and without Outlier Exposure for the threshold calibration dataset we used in Section 4.3.2. Without Outlier Exposure, there are still a lot of out-of-distribution examples with large softmax scores (0.5 to 1).

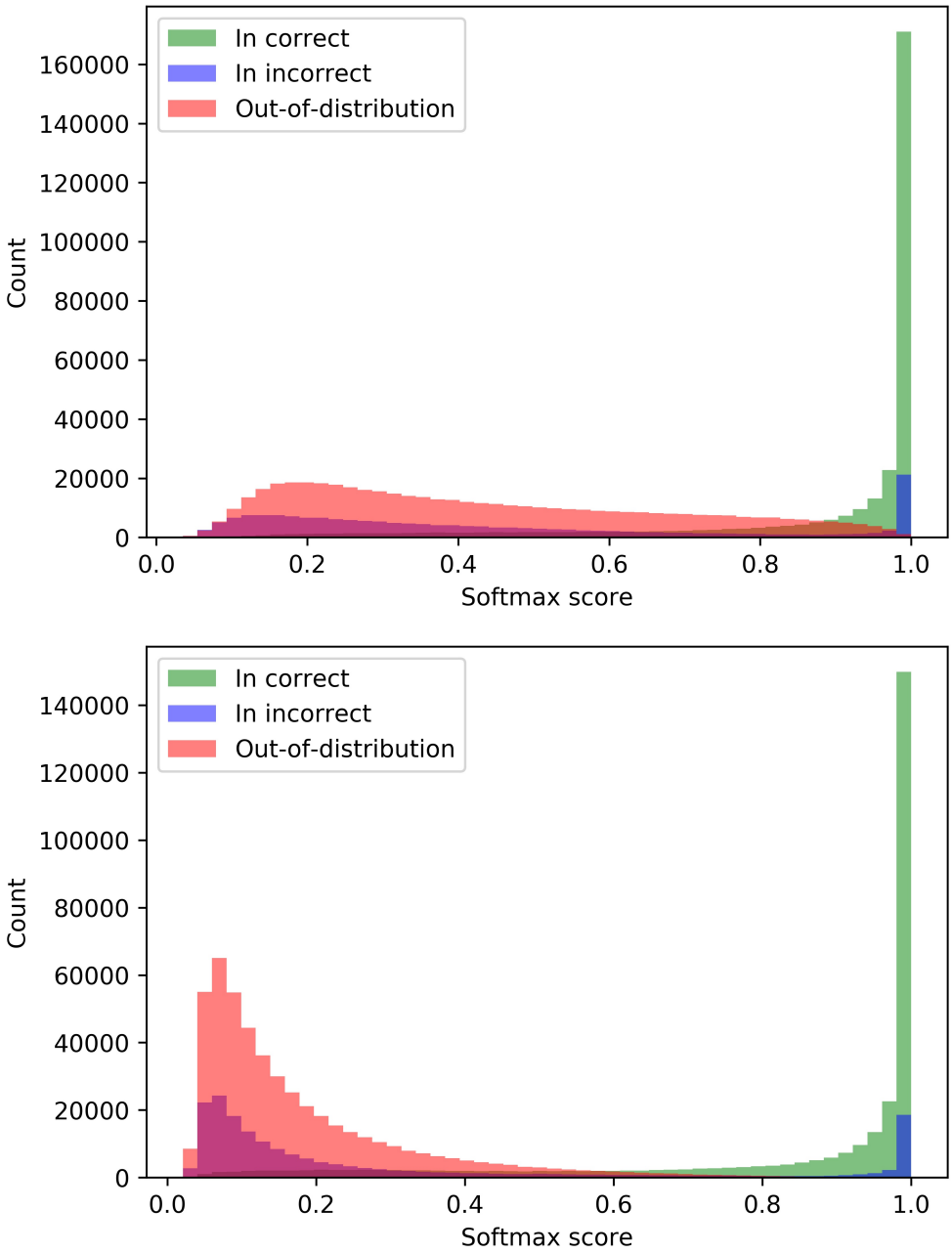

Figure 4.7: The distribution of softmax scores from base model (**top**) and model with Outlier Exposure (**bottom**). Green bar represent the count of in-distribution examples that predicted correctly. Blue bar represent the count of in-distribution examples that predicted incorrectly. Red bar represent the count of out-of-distribution examples.

After trained with Outlier Exposure, most of the out-of-distribution examples accumulated with small softmax scores (0 to 0.4). We also found that the scores of in-distribution examples that incorrectly predicted by our model are close to the scores of out-of-distribution examples. So an optimized threshold can also exclude those incorrect predictions.

4.2.4 Implementations and hyperparameters

DeepFrame was implemented by Python3 using PyTorch [119] with Apex [111] acceleration. The DeepFrame model was trained using a gradient-descent optimization algorithm with adaptive estimates of moments called Adam [75]. The training target is to minimize multiple class cross-entropy loss function.

We also implemented several commonly practiced techniques for optimizing deep neural networks, such as mini-batch gradient descent, He initialization [61], and hyperparameter tuning. There are several hyperparameters including the number and the size of convolution filters, the number of input and output features in the linear layer, dropout rate, learning rate, and batch size. We tried our best to search the parameters, and we used the parameters that are tested to perform well in the experiment. In our experiment, the hyperparameters were set as follows: batch size $= 256$, learning rate $= 0.001$, dropout rate $= 0.5$. For the first convolutional layer, the size of the filters is 3, and the number of filters is 64. For the second convolutional layer, the size of the filters is $8, 12, 16, 20, 24, 28, 32, 36$ with 256 filters of each size. Also, for the hidden linear layer, the number of output features is 512. Given the data size we used, we found our model converges around 1 to 2 epochs training, so we trained all models two epochs with validating every 102400 samples to select the best model.

4.3 Experiments and results

To evaluate DeepFrame model, we applied DeepFrame on two dataset: a simulated PacBio G protein-coupled receptor (GPCR) coding sequences (cds) dataset [30], and a real third-generation sequencing dataset of human genome [20, 116]. G protein-coupled receptor is a large protein family that is involved in many critical physiological processes, such like visual sense, gustatory sense, sense of smell, regulation of immune system activity, and so on [143]. We used GDS [30] for our experiment as it is already used to evaluate DeepFam [133]. It consist of 8222 protein sequences belonging to 5 families, 38 subfamilies, and 86 sub-subfamilies. For simulated GPCR cds dataset, we have the protein family information of reference read for each simulated read as our ground truth. For the human genome dataset, we determine the ground truth via BLASR's [21] alignments against the GPCR coding sequences.

We benchmarked DeepFrame's performance with HMMER and DeepFam. Those tools and methods are representatives of current state-of-the-art methods of protein domain prediction. In experiments on simulated PacBio GPCR cds dataset, we evaluate the performance of all methods using classification accuracy. For human genome dataset, we evaluated the performance of detection of GPCR protein families from other unrelated DNA sequences using following metrics: (1) recall, which measures the ratio of the correct protein domains predicted by each program to the whole set of expected protein domains; (2) precision, which quantifies the ratio of correct protein domains detected by each program to the total reported protein domains; and (3) F1 score, which is the harmonic mean of recall and precision. Because DeepFam is not designed to process the data with out-of-distribution samples, we did not evaluate it in the experiment on the human genome dataset.

For our experiments, all specific commands, parameters, and output can be found along with

the source code of DeepFrame.

4.3.1 Simulated PacBio GPCR cds dataset

We first evaluated our model using simulated PacBio reads generated using PBSIM [114] with GPCR cds dataset with default setup and error rates from 1% to 15%. We create a corresponding GPCR protein-coding sequences (cds) dataset by downloading cds from NCBI by searching the keyword of the corresponding sub-subfamilies in GDS. We also build pHMM models from GDS using HMMER and test all cds against these models. We discarded the cds that not predicted correctly by GPCR HMM models to clean the dataset. We determined the ground truth of sub-subfamily label of a simulated read by assign the label of the reference coding sequence to it.

4.3.1.1 Performance with different architectures

We conducted a series of experiments by varying the key opponents in our base models: the number of convolution layers, the number of convolution kernels, the size of convolution kernels. We also tested different encoding strategies. We listed all variations we tested in Table 4.1. The reference coding sequences of each sub-subfamilies were spitted to 80% training samples and 20% test samples. Then we used PBSIM to generate simulated PacBio reads for training samples and test samples with 15% error rates. All the models were trained using the same hyperparameters we discussed in Section 4.2.4 with five times repeat. Figure 4.8 compares classification accuracy of all the variants in the test samples of simulated GPCR cds dataset.

4.3.1.1.1 Comparisons of encoding. We compared three models with different encoding strategies: (1) 3-frame encoding model, (2) DNA one-hot encoding model, and (3) 3-branch model. We describe the architecture of all three encoding models in Section 4.2.1.

Name	Architecture (compare to base model)
Base model	The model we described in Methods
512filters	Use 512 filters in total in the 2nd convolution layer
1024filters	Use 1024 filters in total in the 2nd convolution layer
4096filters	Use 4096 filters in total in the 2nd convolution layer
1layer	Only keep the last convolution layer
3layer	Add an extra convolution layer with 64 filters of size 3
filters6	filter sizes of 2nd convolution layer= $[6, 9, 12, 15, 18, 21, 24, 27]$
filters10	filter sizes of 2nd convolution layer= $[10, 15, 20, 25, 30, 35, 40, 45]$
filters12	filter sizes of 2nd convolution layer= $[12, 18, 24, 30, 36, 42, 48, 54]$
3-branch	3 branches structure for translated reads
DNA-encoding	Use one-hot encoding of DNA reads as input

Table 4.1: The name and brief description of variants of CNN models we compared.

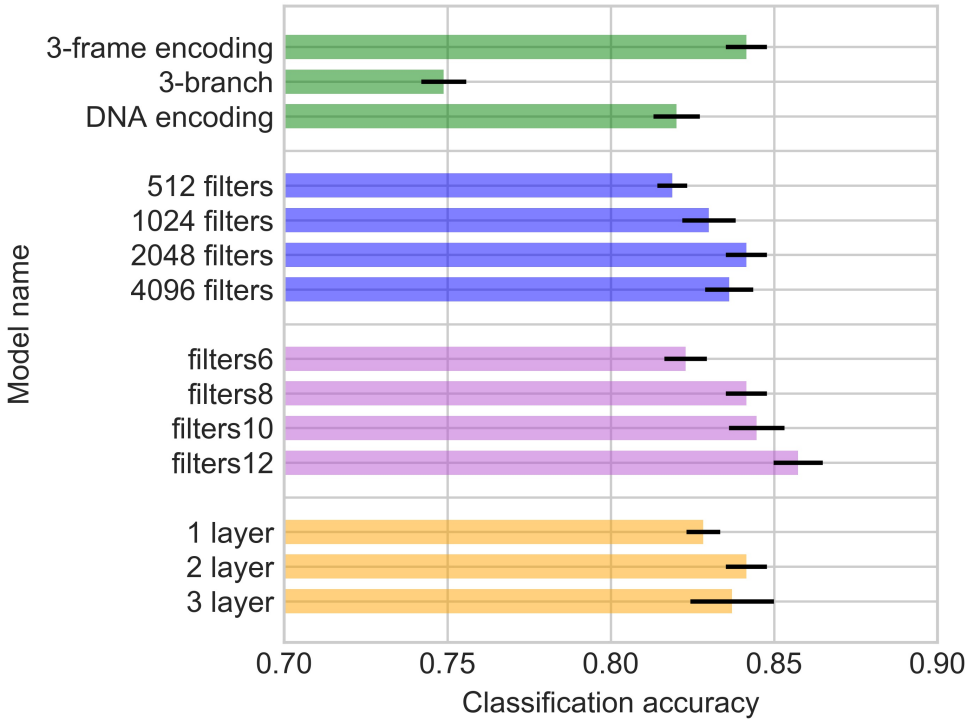

Figure 4.8: The mean and standard deviation of classification accuracy of different network architectures. For convenient, we used different names (*3-frame encoding*, *2048 filters*, *filters8*, and *2 layer*) for the same base model. We used different colors for different group of comparisons: green bars for encoding; blue bars for number of filters; purple bars for different filter sizes; orange bars for different convolution layers.

From Figure 4.1, the model with 3-frame encoding achieved much better performance compared to the 3-branch model and DNA encoding. In the 3-branch model, the original DNA sequences information is lost as the orders of the three frames not kept in the model. In contrast, DNA encoding has the original information, but the indirect information of protein in DNA is not easy to extract useful features that can separate different protein families. The DNA encoding model also converges much slower than the other two peptide encoding models in training. Finally, both DNA encoding model and 3-branch model have a larger number of parameters compared to 3-frame encoding, requiring more computing resources.

To investigate the features extracted from the sequences by convolutional layers, we obtain the feature matrix after the max-over-time pooling for all three models. We pick six sub-subfamilies with top F1 scores for all three models for visualizing the features using t-SNE [96]. The features from different protein sub-subfamilies have been mapper to many small clusters shown in 4.9. The distances between clusters from different sub-subfamilies in 3-frame encoding models are relatively larger, and each small cluster only represents one sub-subfamilies. Many clusters in DNA one-hot encoding model are close to each other. And clusters from the 3-branch model have overlaps. These plots may explain why the performance of the 3-frame encoding is better as it extracts features that can easily distinguish different protein sub-subfamilies.

4.3.1.1.2 Comparisons of convolutional filters setup. Compared to the one-layer model, our base model (2 layers) achieved higher accuracy. The extra layer helps the neural network to extract more complex patterns such as interactions of the lower level features. However, the "deeper" model with more layers are more difficult to optimize. That is the possible reason why the average accuracy of 3 layer model is lower than the base model, but the highest accuracy achieved is higher than our base model.

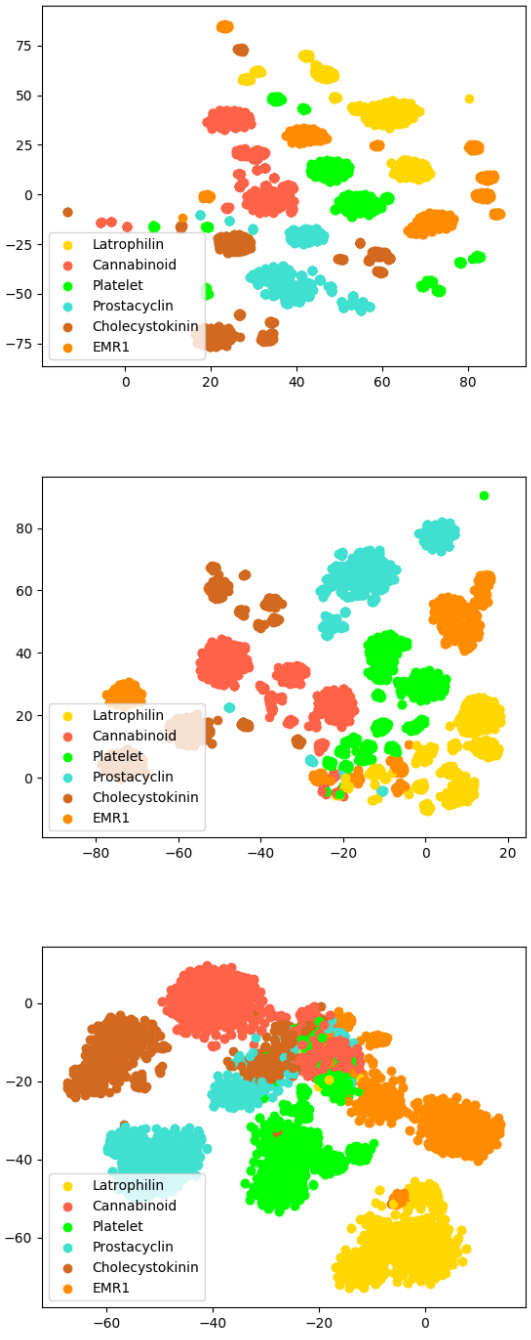

Figure 4.9: 2D t-SNE plot of features extracted from the convolutional layer output. (Top) 3-frame encoding model. (Middle) DNA one-hot encoding model. (Bottom) 3-branch model.

We found that the additional convolutional filters increased performance for protein domain prediction. The improvement is saturated when we have more than 2048 filters.

Increasing the size of filters can also help to improve the performance of the models. With the larger filter size, the neural network can capture long-range features. The result suggests the importance of choosing the right filter size, which is not explored in previous works [133, 153].

4.3.1.2 Comparison with HMMER and DeepFam

The accuracy of classification of GPCR cds of DeepFrame and DeepFam was measured using 5-fold cross-validation. The reference coding sequences of each sub-subfamilies were equally split into five folds while preserving the ratio of the families. Then we use PBSIM simulate each fold of the coding sequences to the simulated PacBio reads that used to train DeepFrame. For DeepFam, we used the translated protein sequences of each fold of coding sequences to retrain it. We then tested all three translations using our retrained model. For HMMER, we used all 5-fold translated protein sequences to retrain the pHMM model. To generate multiple seuqnce aignment, MAFFT [72] was used for the sequences of each sub-subfamilies. Then we used `hmmbuild` in HMMER to build pHMM model. For each test DNA sequences, 3-frame translations was applied to get three protein sequences, then was tested with `hmmscan` against all 86 pHMM models we built. In both experiments of DeepFam and HMMER, if at least one of the three translated reads classified to the right sub-subfamilies, we treated this case as a correct prediction. HMMER was not able to assign family label for some sequences, so such cases were treated as incorrect predictions.

Figure 4.10 summarized the comparison of classification of all methods on the simulated PacBio reads. For this data set, our method achieved consistent performance when test on dataset with different error rates. Our method achieved much higher accuracy compared to the other two methods when the error rates are high. The high error rates heavily impacted the performance of

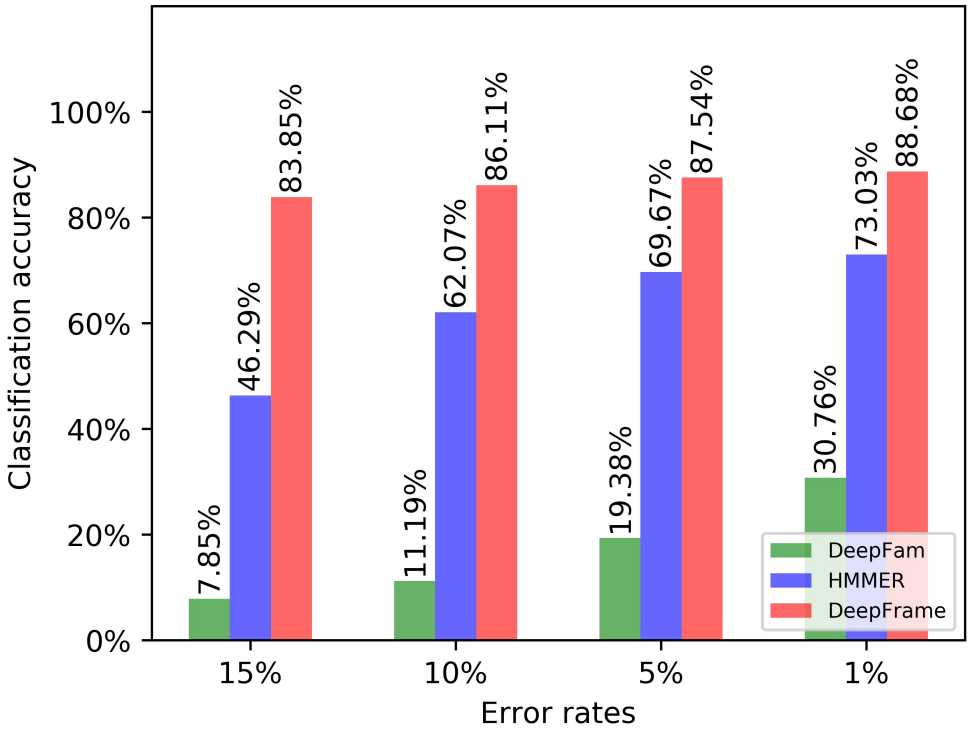

Figure 4.10: Comparison of protein classification performance of DeepFrame, HMMER, and DeepFam across different error rates.

HMMER and DeepFam. It is expected because the profile HMM search is much more sensitive to frameshifts caused by gaps, and DeepFam is designed for classifying relatively complete error-free protein sequences.

4.3.1.3 Comparison of time complexity

We measured the execution times of the tested methods. With a large amount of data generated by the third-generation sequencing platform, we require not only high accuracy of the algorithm but also the high efficiency of the algorithms. We run both DeepFrame, DeepFam, and HMMER using Intel® Xeon® Gold 6148 CPU with 20 cores at the High-Performance Computing Center at Michigan State University. We also tested DeepFrame and DeepFam with NVIDIA® Tesla® V100 GPU with Apex acceleration library (HMMER doesn't support GPU). For each method, We measured its execution time by averaging 5 independent trials with randomly selected 10,000 sequences.

Setup	DeepFrame	DeepFam	HMMER	HMMER `--max`
CPU	1168.78s	276.74s	312.13s	3470.04s
GPU	25.71s	20.37s	unavailable	unavailable

Table 4.2: The average elapsed time to predict families of 10,000 simulated PacBio reads for each method.

In CPU, HMMER with the default setup runs much faster than DeepFrame and DeepFam. With high sequencing error rates, the alignment against many candidate sub-subfamilies cannot pass the filter stage of HMMER, speeding up pHMM search. With `--max` (turn off filters), the execution time of HMMER is even slower than deep learning methods. With GPU acceleration, the running time of DeepFrame is much smaller than the running time of HMMER with the default setup.

4.3.2 Human genome dataset

To evaluate DeepFrame's performance on real third-generation sequencing dataset, we tested DeepFrame on the H. sapiens 10x Sequence Coverage with PacBio data [116] and Oxford Nanopore Human Reference Datasets Rel6 [67]. We first described the dataset we used in this experiment.

4.3.2.0.1 Train dataset. For the threshold baseline, we used the model trained in the previous classification experiment. To applying Outlier Exposure, we constructed a dataset that mixing the previous 5-fold train dataset with a dataset of outliers. We constructed the outelier dataset using simulated PacBio reads. To close to the real out-of-distribution examples, we simulated a PacBio human genome dataset using GRCh37/hg19 human reference genome as reference [27]. We only kept the simulated reads that cannot be aligned to cds by BLASR in the final out-of-distribution dataset.

4.3.2.0.2 Threshold calibration dataset. To calibrate the threshold, we need a dataset that mixing in-distribution examples with out-of-distribution examples. We simulated in-distribution examples from GPCR cds dataset. Also, we simulated out-of-distribution examples using ATP synthase families cds. We used ATP synthase families because both GPCR and ATP synthase families belong to the membrane and cell surface proteins and peptides class in SCOP classification [92]. Intuitively, protein families closely related may be more challenging to distinguish, and this lead to the threshold we calibrated more robust.

4.3.2.0.3 Out-of-distribution test dataset. We constructed two test dataset: a PacBio RS II test dataset from PacBio SMRT Sequencing for CHM1TERT human cell line; and a Nanopore test dataset from Oxford Nanopore MinION on CEPH1463 (NA12878/GM12878, Ceph/Utah pedigree) human genome reference standard using R9.4 chemistry. To determine the ground truth of

these test reads, we first aligned all reads against GPCR cds dataset using BLASR [21]. We extracted the aligned part of the reads of which alignment length longer than 60% of aligned cds length as our in-distribution test samples. For each sample, the ground truth is given by the label of aligned cds. These reads are in-distribution examples in the test dataset. We also randomly selected reads that not aligned to any cds as our out-of-distribution samples. The dataset consists of 50% reads that may be coding GPCR and 50% out-of-distribution reads. For read longer than 3000 bps in the test dataset, we cropped read into fragments that not longer than 3000 bps to be consistent with train dataset.

4.3.2.1 Out-of-distribution test using PacBio and Nanopore reads

Method	PacBio			Nanopore		
	Recall	Precision	F-1 score	Recall	Precision	F-1 score
HMMER	0.1494	0.9507	0.2583	0.3984	**0.9825**	0.5670
DeepFrame threshold	0.4235	0.5407	0.4667	**0.4866**	0.6234	0.5384
DeepFrame OE	**0.4479**	**0.9837**	**0.6154**	0.4836	0.9731	**0.6458**

Table 4.3: The performance of protein domain prediction with out-of-distribution examples using DeepFrame threshold baseline, DeepFrame with Outlier Exposure (OE), and HMMER on the real PacBio and Nanopore dataset.

For detect the GPCR coding sequence sample from out-of-distribution samples, we retrain DeepFrame model with Outlier Exposure discussed in Methods. We used the same train dataset in the previous simulated PacBio reads experiment. We benchmarked the OE model with HMMER and a baseline model with only a softmax threshold. In both PacBio and Nanopore dataset, DeepFrame with OE achieves the best F1 score compared to HMMER default setup and threshold baseline (Table 4.3). DeepFrame with OE achieved significant improvement on recall while the precision is comparable with HMMER. In general, all three methods have better performance on

Nanopore dataset. Noticed that in the whole pipeline, we only use simulated PacBio reads but no data from Nanopore. This result suggests that our strategy is robust with different types of long reads.

4.3.3 Visualize and understanding convolution filters

To better understand what our convolutional neural networks learned, we further investigate the feature extracted by convolutional filters. We visualized the convolution units of our model and found that some conserved regions were captured, although the input data is very noisy.

Following the methods adopted by previous research [3, 133], we visualize the convolution units activated for each family. We used the model that was trained in previously experiments and feed the test sequences that belonged to the family to our model. Then we collect all the sequence fragments that activate the convolution units. We extracted the results from most activated convolution units and used Weblogo to generate the logos from sub-sequences of the results. Since we translated the original input DNA sequences into 3-frames protein sequences, so for each convolutional units, we have 3 logos associated with 3 frames respectively.

From the logos, we can found that the convolutional filters did learn some conserved regions of protein families. However, the logo is very noisy compared to the results reported in previous studies. When the input has insertions and deletions, a well-conserved region may appear in all three frames due to frameshifts, so the filter of all three channels sometimes may need to extract similar patterns. That is why some of the 3-frame logos show similar patterns across different frames.

Figure 4.17 presented the activation value distribution of convolutional filters for the six sub-subfamilies we discussed before. It clearly shows that different protein families will activate different convolutional filters.

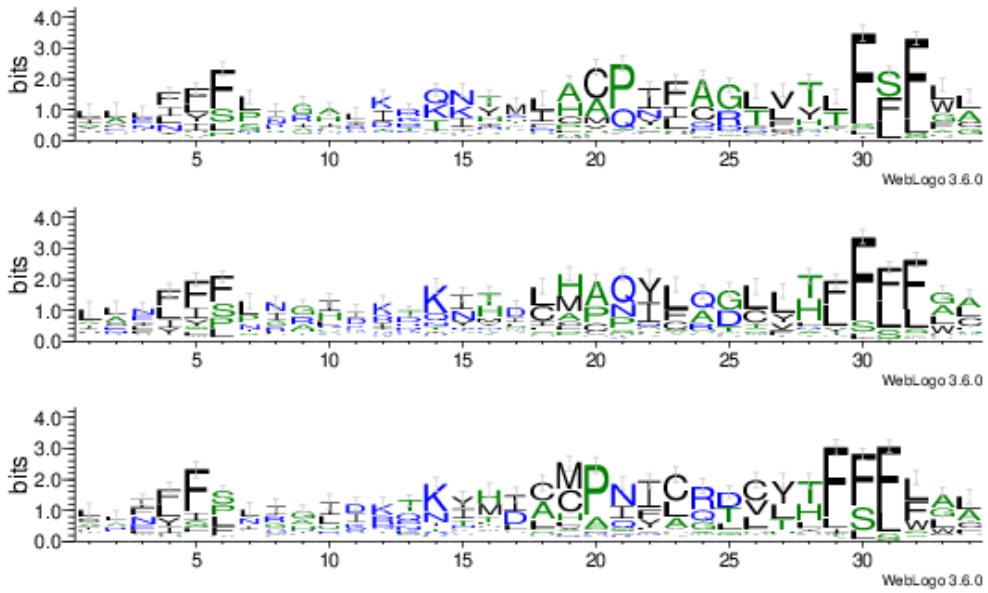

Figure 4.11: Most activated convolutional filters (index: 1622) for Latrophilin.

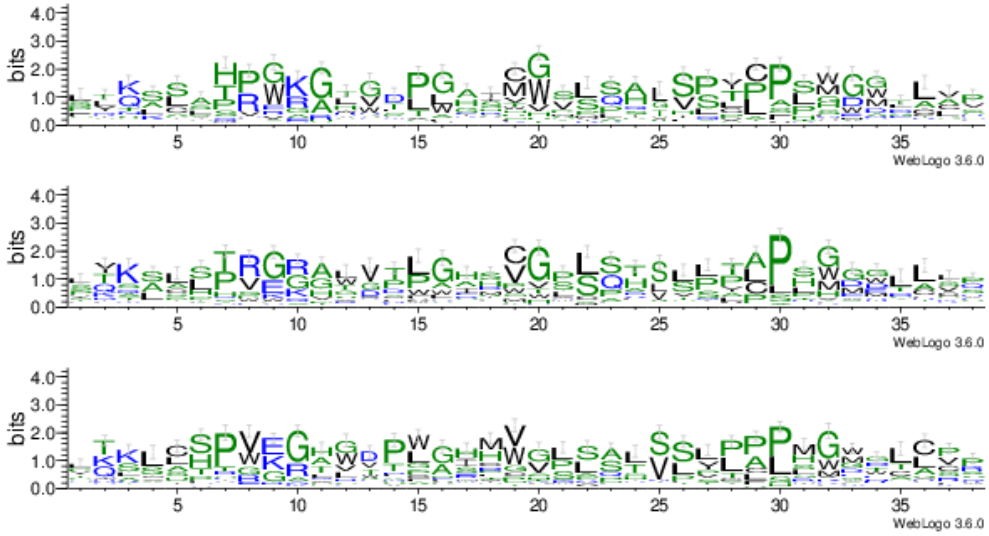

Figure 4.12: Most activated convolutional filters (index: 1877) for Cannabinoid.

4.4 Discussion

In this work, we adopted the threshold on softmax and Outlier Exposure to detect relevant target reads from other random reads. Previous research [3, 65, 91] usually solve the challenge by using a binary classification framework. However, for a given sequence, we need to run the binary classification model for each possible target. Instead, our multi-classification model scales better

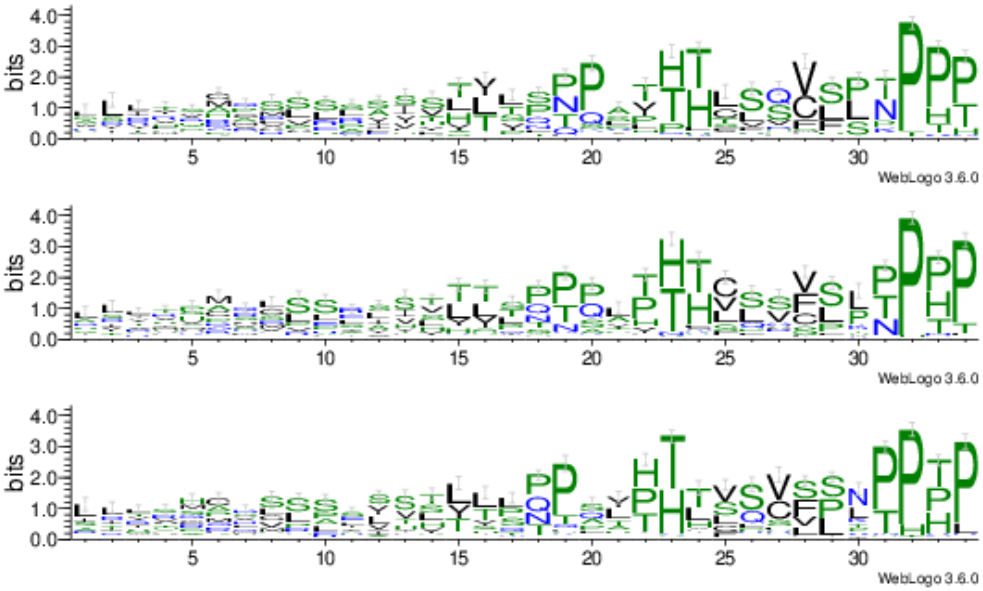

Figure 4.13: Most activated convolutional filters (index: 1689) for Platelet.

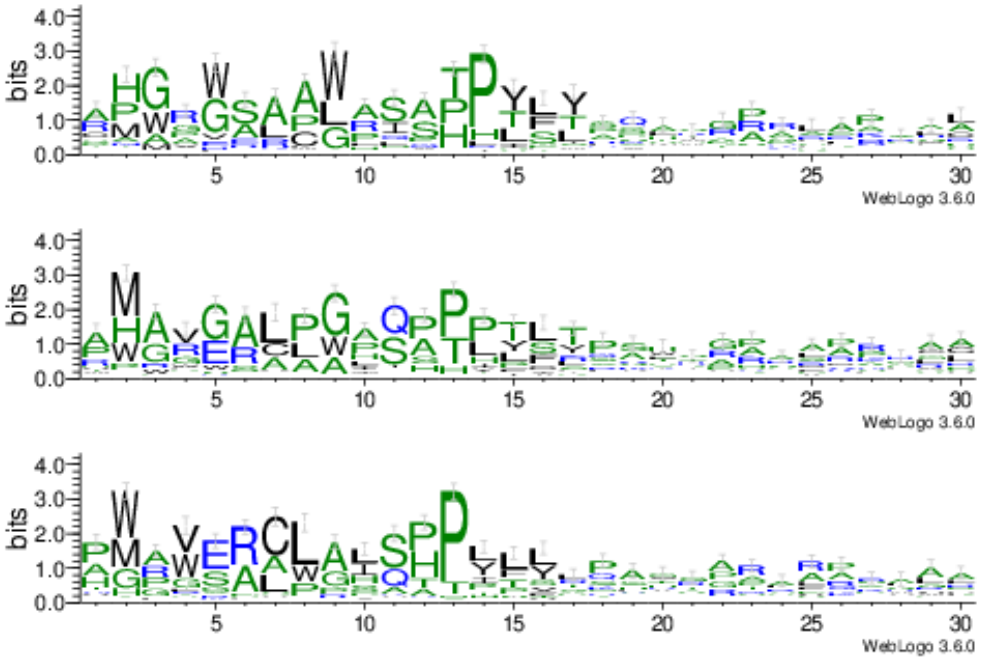

Figure 4.14: Most activated convolutional filters (index: 1382) for Prostacyclin.

with a large number of candidate targets.

We addressed the challenge of prediction of protein domains with a supervised learning method, which needs a vast amount of labeled data. However, it is challenging to obtain a large number of

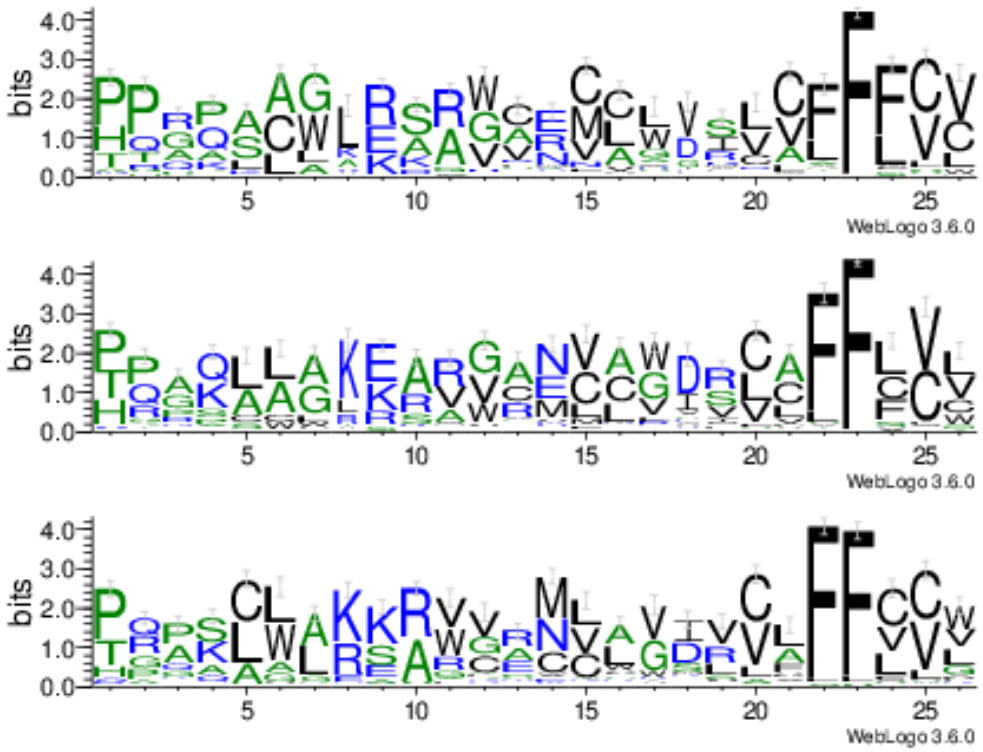

Figure 4.15: Most activated convolutional filters (index: 1033) for Cholecystokinin.

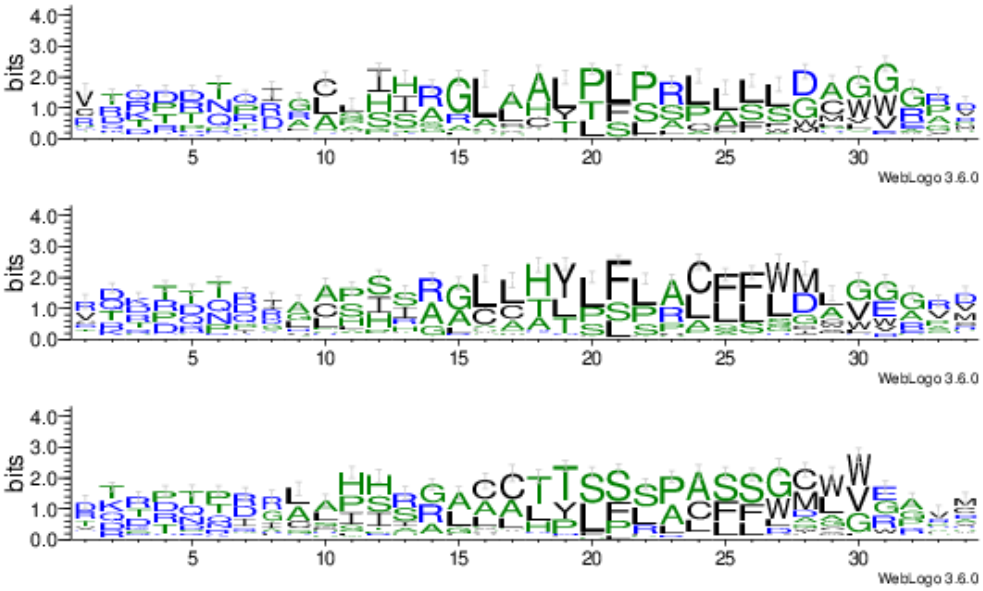

Figure 4.16: Most activated convolutional filters (index: 1776) for EMR1.

labeled reads from third-generation sequencing. In this work, we show that deep learning method achieves acceptable results, even when training on simulated data and testing on real data. We be-

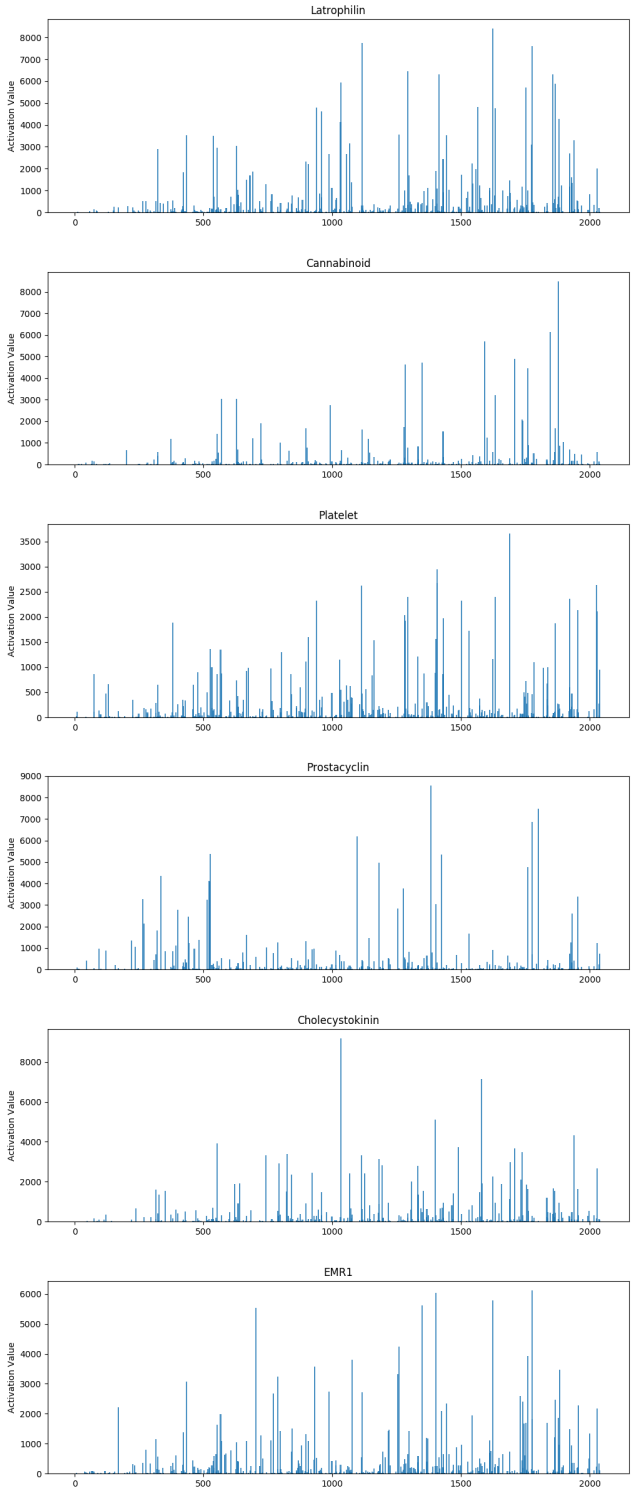

Figure 4.17: The distributions of sum of acivation values of convolutional units. X axis is the index of the convolution filters in DeepFram. Y axis is the sum of activation values of the given convolutional filters.

lieve the performance of our method can be further improved if more labeled real third-generation reads available.

4.5 Conclusion

In this work, we developed a protein domain prediction tool for third-generation sequencing data. By incorporating 3-frame encoding and multiple-layer convolutional filters, our CNN model can directly predict the protein families of long erroneous read. Our experimental results have shown that for the dataset from third-generation sequencing technologies, our program can detect relevant protein domain from other random DNA reads. Being able to predict protein domain from a single read from third-generation sequencing directly is essential for applications like transcriptomic and metagenomic sequencing. DeepFrame provides a complementary tool to current third-generation sequence analysis pipelines, which usually require high coverage for error correction. The source code and the models trained and configuration files are available at https://github.com/strideradu/DeepFrame.

Chapter 5

Conclusions and future works

In this dissertation, I first introduced the development of third-generation sequencing technologies and the possible application of PacBio and Nanopore reads. As more and more third-generation sequencing dataset accumulating, especially metagenomic data and transcriptomic data, the analysis of long noisy reads became a challenging problem. First, traditional sequence analysis tools cannot handle high insertion and deletion rates, leading to the short annotation that cannot be distinguished from random sequences. Second, applications like metagenomic sequencing may contain many close protein sequences that are not easy to distinguish. Third, although some algorithms have been developed for PacBio or Nanopore reads, there is still a great potential to improve the performance and efficiency of the sequence analysis algorithm for third-generation reads. To address these challenges, I proposed a set of algorithms in this dissertation.

In Chapter 2, I introduced GroupK, which was designed for the high sensitivity overlap detection of long noisy reads. GroupK incorporated the grouped hits criteria which has been successfully applied to remote homology detection and achieves better sensitivity when compared with other overlap detection tools for third-generation sequencing reads.

In Chapter 3, I introduced Frame-Pro, which was designed for the homology search of PacBio reads. It corrects sequencing errors and also outputs the profile alignments of the corrected sequences against characterized protein families simultaneously. Compared with homology search on error correction reads, Frame-Pro enables more sensitive homology search and corrects more

sequencing errors.

In Chapter 4, I introduce DeepFrame, which was designed for protein domain prediction and detection on a single long noisy read. DeepFrame is based on the deep convolutional neural networks with softmax threshold and Outlier Exposure. It outperforms HMMER on both classification accuracy and detection F1 score with faster execution time using GPU.

There are several directions that the previous studies can be improved:

- In GroupK, the group seed matching step currently is based on hash table implementation from YASS [108]. This step is the bottleneck of the GroupK overlap detection pipeline. In the future, we can implement the group matching step using new k-mer finding method to improve the running time efficiency to be comparable with other faster overlap detection tools. We can also invest in the filter steps to allow less false-positive overlap pairs to enter the downstream steps.

- Frame-Pro is computationally more expensive than HMMER as we do not adopt any acceleration approach. We can reduce the running time of Frame-Pro by pruning the dynamic programming matrix. We can also incorporate the fast Viterbi algorithm as a filtration stage to filter out impossible protein domain candidates. Also, Frame-Pro is currently implemented with Python. The execution time of Frame-Pro can be greatly reduced if we implement it using C++, or other accelerated machine learning libraries like TensorFlow [1], or PyTorch [119].

- In DeepFrame, we implemented two convolutional layer neural networks. In the future, we may test a more complex model with "deeper" convolutional neural networks. However, usually "deeper" neural networks are not easy to optimize, so we may need use techniques like residue blocks, as used in ResNet [62]. Another candidate is transformer structure [146],

which already proved its success on natural language processing [31]. We may also add a filtration step to improve the efficiency of our algorithm further, especially when only CPU available. Besides, currently we use multi-class classification framework that directly classifies on the sub-subfamilies level. We may implement a hierarchical classification framework that classify on different protein levels like families, sub-families, and sub-subfamilies simultaneously.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] S. Aki, H. Kuboki, and K. Hirano. On discrete distributions of orderk. *Annals of the Institute of Statistical Mathematics*, 36(1):431–440, 1984.

[3] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey. Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature biotechnology*, 33(8):831, 2015.

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[5] I. Antonov and M. Borodovsky. Genetack: frameshift identification in protein-coding sequences by the viterbi algorithm. *Journal of bioinformatics and computational biology*, 8(03):535–551, 2010.

[6] Apple Inc. Performing convolution operations. https://developer.apple.com/library/archive/documentation/Performance/Conceptual/vImage/ConvolutionOperations/ConvolutionOperations.html, 2018. Sep, 2016.

[7] S. Ardui, A. Ameur, J. R. Vermeesch, and M. S. Hestand. Single molecule real-time (smrt) sequencing comes of age: applications and utilities for medical diagnostics. *Nucleic acids research*, 46(5):2159–2168, 2018.

[8] K. F. Au, J. G. Underwood, L. Lee, and W. H. Wong. Improving pacbio long read accuracy by short read alignment. *PLoS One*, 7(10):e46679, 2012.

[9] L. Baum. An inequality and associated maximization technique in statistical estimation of probabilistic functions of a markov process. *Inequalities*, 3:1–8, 1972.

[10] A. Bendale and T. E. Boult. Towards open set deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1563–1572, 2016.

[11] G. Benson et al. Tandem repeats finder: a program to analyze dna sequences. *Nucleic acids research*, 27(2):573–580, 1999.

[12] K. Berlin, S. Koren, C.-S. Chin, J. P. Drake, J. M. Landolin, and A. M. Phillippy. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature biotechnology*, 33(6):623–630, 2015.

[13] E. Birney, M. Clamp, and R. Durbin. Genewise and genomewise. *Genome research*, 14(5):988–995, 2004.

[14] M. Borodovsky and J. McIninch. Genmark: parallel gene recognition for both dna strands. *Computers & chemistry*, 17(2):123–133, 1993.

[15] D. Branton, D. W. Deamer, A. Marziali, H. Bayley, S. A. Benner, T. Butler, M. Di Ventra, S. Garaj, A. Hibbs, X. Huang, et al. The potential and challenges of nanopore sequencing. *Nature biotechnology*, 26(10):1146–1153, 2008.

[16] A. Z. Broder. Identifying and filtering near-duplicate documents. In *Annual Symposium on Combinatorial Pattern Matching*, pages 1–10. Springer, 2000.

[17] N. P. Brown, C. Sander, and P. Bork. Frame: detection of genomic sequencing errors. *Bioinformatics (Oxford, England)*, 14(4):367–371, 1998.

[18] J. Buhler, U. Keich, and Y. Sun. Designing seeds for similarity search in genomic DNA. *Journal of Computer and System Sciences*, 70(3):342–363, 2005.

[19] A. Busia, G. E. Dahl, C. Fannjiang, D. H. Alexander, E. Dorfman, R. Poplin, C. Y. McLean, P.-C. Chang, and M. DePristo. A deep learning approach to pattern recognition for short dna sequences. *bioRxiv*, page 353474, 2019.

[20] M. J. Chaisson, J. Huddleston, M. Y. Dennis, P. H. Sudmant, M. Malig, F. Hormozdiari, F. Antonacci, U. Surti, R. Sandstrom, M. Boitano, et al. Resolving the complexity of the human genome using single-molecule sequencing. *Nature*, 517(7536):608–611, 2015.

[21] M. J. Chaisson and G. Tesler. Mapping single molecule sequencing reads using basic local alignment with successive refinement (blasr): application and theory. *BMC bioinformatics*, 13(1):238, 2012.

[22] W. I. Chang and E. Lawler. Sublinear expected time approximate string matching and biological applications. *Algorithmica*, 12:327–44, 1994.

[23] T. Charalampous, G. L. Kay, H. Richardson, A. Aydin, R. Baldan, C. Jeanes, D. Rae, S. Grundy, D. J. Turner, J. Wain, et al. Nanopore metagenomics enables rapid clinical diagnosis of bacterial lower respiratory infection. *Nature Biotechnology*, page 1, 2019.

[24] C.-S. Chin, D. H. Alexander, P. Marks, A. A. Klammer, J. Drake, C. Heiner, A. Clum, A. Copeland, J. Huddleston, E. E. Eichler, et al. Nonhybrid, finished microbial genome assemblies from long-read smrt sequencing data. *Nature methods*, 10(6):563–569, 2013.

[25] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[26] J. Chu, H. Mohamadi, R. L. Warren, C. Yang, and I. Birol. Innovations and challenges in detecting long read overlaps: an evaluation of the state-of-the-art. *Bioinformatics*, 33(8):1261–1270, 2016.

[27] D. M. Church, V. A. Schneider, T. Graves, K. Auger, F. Cunningham, N. Bouk, H.-C. Chen, R. Agarwala, W. M. McLaren, G. R. Ritchie, et al. Modernizing reference genome assemblies. *PLoS biology*, 9(7):e1001091, 2011.

[28] J. Clarke, H.-C. Wu, L. Jayasinghe, A. Patel, S. Reid, and H. Bayley. Continuous base identification for single-molecule nanopore dna sequencing. *Nature nanotechnology*, 4(4):265, 2009.

[29] P. Compeau and P. Pevzner. *Bioinformatics algorithms: an active learning approach*, volume 1. Active Learning Publishers, La Jolla, California, USA, 2015.

[30] M. N. Davies, A. Secker, A. A. Freitas, M. Mendao, J. Timmis, and D. R. Flower. On the hierarchical classification of g protein-coupled receptors. *Bioinformatics*, 23(23):3113–3118, 2007.

[31] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[32] A. Döring, D. Weese, T. Rausch, and K. Reinert. SeqAn an efficient, generic c++ library for sequence analysis. *BMC bioinformatics*, 9(1):11, 2008.

[33] N. Du, J. Chen, and Y. Sun. Improving the sensitivity of long read overlap detection using grouped short k-mer matches. *BMC genomics*, 20(2):190, 2019.

[34] N. Du and Y. Sun. Improve homology search sensitivity of PacBio data by correcting frameshifts. *Bioinformatics*, 32(17):i529–i537, 2016.

[35] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, Cambridge, United Kingdom, 1998.

[36] S. R. Eddy. Profile hidden markov models. *Bioinformatics (Oxford, England)*, 14(9):755–763, 1998.

[37] S. R. Eddy, T. J. Wheeler, and the HMMER development team. Hmmer 3.1b2, 2015.

[38] S. R. Eddy, T. J. Wheeler, and the HMMER development team. Hmmer 3.2.1. http:

//hmmer.org/, 2018. June, 2018.

[39] J. Eid, A. Fehr, J. Gray, K. Luong, J. Lyle, G. Otto, P. Peluso, D. Rank, P. Baybayan, B. Bettman, et al. Real-time dna sequencing from single polymerase molecules. *Science*, 323(5910):133–138, 2009.

[40] S. El-Gebali, J. Mistry, A. Bateman, S. R. Eddy, A. Luciani, S. C. Potter, M. Qureshi, L. J. Richardson, G. A. Salazar, A. Smart, et al. The pfam protein families database in 2019. *Nucleic acids research*, 47(D1):D427–D432, 2018.

[41] W. Feller. *An introduction to probability theory and its applications*, volume 2. John Wiley & Sons, 2008.

[42] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE, 2000.

[43] A. Fiannaca, L. La Paglia, M. La Rosa, G. Renda, R. Rizzo, S. Gaglio, A. Urso, et al. Deep learning models for bacteria taxonomic classification of metagenomic data. *BMC bioinformatics*, 19(7):198, 2018.

[44] R. D. Finn, A. Bateman, J. Clements, P. Coggill, R. Y. Eberhardt, S. R. Eddy, A. Heger, K. Hetherington, L. Holm, J. Mistry, et al. Pfam: the protein families database. *Nucleic acids research*, 42(D1):D222–D230, 2013.

[45] R. D. Finn, P. Coggill, R. Y. Eberhardt, S. R. Eddy, J. Mistry, A. L. Mitchell, S. C. Potter, M. Punta, M. Qureshi, A. Sangrador-Vegas, et al. The pfam protein families database: towards a more sustainable future. *Nucleic acids research*, 44(D1):D279–D285, 2015.

[46] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[47] Z. Gao, C.-h. Tseng, Z. Pei, and M. J. Blaser. Molecular analysis of human forearm superficial skin bacterial biota. *Proceedings of the National Academy of Sciences*, 104(8):2927–2932, 2007.

[48] F. Giordano, L. Aigrain, M. A. Quail, P. Coupland, J. K. Bonfield, R. M. Davies, G. Tischler, D. K. Jackson, T. M. Keane, J. Li, et al. De novo yeast genome assemblies from minion, pacbio and miseq platforms. *Scientific reports*, 7(1):3935, 2017.

[49] M. Gîrdea, L. Noé, and G. Kucherov. Back-translation for discovering distant protein homologies. In *International Workshop on Algorithms in Bioinformatics*, pages 108–120. Springer, 2009.

[50] M. Gîrdea, L. Noé, and G. Kucherov. Back-translation for discovering distant protein homologies in the presence of frameshift mutations. *Algorithms for Molecular Biology*, 5(1):6, 2010.

[51] G. Gonnella and S. Kurtz. Readjoiner: a fast and memory efficient string graph-based sequence assembler. *BMC bioinformatics*, 13(1):82, 2012.

[52] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, Cambridge, Massachusetts, United States, 2016.

[53] X. Guan and E. C. Uberbacher. Alignments of dna and protein sequences containing frameshift errors. *Bioinformatics*, 12(1):31–40, 1996.

[54] D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge university press, 1997.

[55] T. Hackl, R. Hedrich, J. Schultz, and F. Förster. proovread: large-scale high-accuracy pacbio correction through iterative short read consensus. *Bioinformatics*, 30(21):3004–3011, 2014.

[56] D. H. Haft, B. J. Loftus, D. L. Richardson, F. Yang, J. A. Eisen, I. T. Paulsen, and O. White. Tigrfams: a protein family resource for the functional identification of proteins. *Nucleic acids research*, 29(1):41–43, 2001.

[57] D. H. Haft, J. D. Selengut, and O. White. The tigrfams database of protein families. *Nucleic acids research*, 31(1):371–373, 2003.

[58] E. Haghshenas, F. Hach, S. C. Sahinalp, and C. Chauve. Colormap: Correcting long reads by mapping short reads. *Bioinformatics*, 32(17):i545–i551, 2016.

[59] E. Halperin, S. Faigler, and R. Gill-More. Frameplus: aligning dna to protein sequences. *Bioinformatics*, 15(11):867–873, 1999.

[60] K. Hayashi, N. Morooka, Y. Yamamoto, K. Fujita, K. Isono, S. Choi, E. Ohtsubo, T. Baba, B. L. Wanner, H. Mori, et al. Highly accurate genome sequences of escherichia coli k-12 strains mg1655 and w3110. *Molecular systems biology*, 2(1), 2006.

[61] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[62] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[63] D. Hendrycks and K. Gimpel. A baseline for detecting misclassified and out-of-distribution

examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[64] D. Hendrycks, M. Mazeika, and T. G. Dietterich. Deep anomaly detection with outlier exposure. *arXiv preprint arXiv:1812.04606*, 2018.

[65] S. Hochreiter, M. Heusel, and K. Obermayer. Fast model-based protein homology detection without alignment. *Bioinformatics*, 23(14):1728–1736, 2007.

[66] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[67] M. Jain, S. Koren, K. H. Miga, J. Quick, A. C. Rand, T. A. Sasani, J. R. Tyson, A. D. Beggs, A. T. Dilthey, I. T. Fiddes, et al. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature biotechnology*, 36(4):338, 2018.

[68] D. Joseph, J. Meidanis, and P. Tiwari. Determining dna sequence similarity using maximum independent set algorithms for interval graphs. In *Scandinavian Workshop on Algorithm Theory*, pages 326–337. Springer, 1992.

[69] D. Jurafsky and J. H. Martin. *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.

[70] J. Kärkkäinen and P. Sanders. Simple linear work suffix array construction. In *International Colloquium on Automata, Languages, and Programming*, pages 943–955. Springer, 2003.

[71] T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Annual Symposium on Combinatorial Pattern Matching*, pages 181–192. Springer, 2001.

[72] K. Katoh and D. M. Standley. Mafft multiple sequence alignment software version 7: improvements in performance and usability. *Molecular biology and evolution*, 30(4):772–780, 2013.

[73] W. J. Kent. Blat—the blast-like alignment tool. *Genome research*, 12(4):656–664, 2002.

[74] Y. Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[75] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[76] A. Kislyuk, A. Lomsadze, A. L. Lapidus, and M. Borodovsky. Frameshift detection in prokaryotic genomic sequences. *International journal of bioinformatics research and applications*, 5(4):458–477, 2009.

[77] M. Kokot, M. Długosz, and S. Deorowicz. Kmc 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, 2017.

[78] S. Koren, M. C. Schatz, B. P. Walenz, J. Martin, J. T. Howard, G. Ganapathy, Z. Wang, D. A. Rasko, W. R. McCombie, E. D. Jarvis, et al. Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nature biotechnology*, 30(7):693–700, 2012.

[79] S. Koren, B. P. Walenz, K. Berlin, J. R. Miller, N. H. Bergman, and A. M. Phillippy. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome research*, 27(5):722–736, 2017.

[80] J. Korlach. Understanding accuracy in smrt® sequencing, 2013.

[81] J. Korlach and S. W. Turner. Zero-mode waveguides. In *Encyclopedia of Biophysics*, pages 2793–2795. Springer, 2013.

[82] A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler. Hidden markov models in computational biology: Applications to protein modeling. *Journal of molecular biology*, 235(5):1501–1531, 1994.

[83] B. Langmead and S. L. Salzberg. Fast gapped-read alignment with bowtie 2. *Nature methods*, 9(4):357–359, 2012.

[84] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome biology*, 10(3):1, 2009.

[85] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[86] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[87] H. Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, 2016.

[88] H. Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 1:7, 2018.

[89] H. Li and R. Durbin. Fast and accurate long-read alignment with burrows–wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

[90] H. Li and N. Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in bioinformatics*, 11(5):473–483, 2010.

[91] S. Li, J. Chen, and B. Liu. Protein remote homology detection based on bidirectional long short-term memory. *BMC bioinformatics*, 18(1):443, 2017.

[92] L. Lo Conte, B. Ailey, T. J. Hubbard, S. E. Brenner, A. G. Murzin, and C. Chothia. Scop: a structural classification of proteins database. *Nucleic acids research*, 28(1):257–259, 2000.

[93] N. J. Loman, J. Quick, and J. T. Simpson. A complete bacterial genome assembled de novo using only nanopore sequencing data. *Nature methods*, 2015.

[94] B. Ma and M. Li. On the complexity of the spaced seeds. *Journal of Computer and System Sciences*, 73(7):1024–1034, 2007.

[95] B. Ma, J. Tromp, and M. Li. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, 2002.

[96] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[97] U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. *siam Journal on Computing*, 22(5):935–948, 1993.

[98] F. Meyer, R. Overbeek, and A. Rodriguez. Figfams: yet another set of protein families. *Nucleic acids research*, 37(20):6643–6654, 2009.

[99] G. Miclotte, M. Heydari, P. Demeester, S. Rombauts, Y. Van de Peer, P. Audenaert, and J. Fostier. Jabba: hybrid error correction for long sequencing reads. *Algorithms for Molecular Biology*, 11(1):1, 2016.

[100] E. W. Myers. The fragment assembly string graph. *Bioinformatics*, 21(suppl_2):ii79–ii85, 2005.

[101] E. W. Myers, G. G. Sutton, A. L. Delcher, I. M. Dew, D. P. Fasulo, M. J. Flanigan, S. A. Kravitz, C. M. Mobarry, K. H. Reinert, K. A. Remington, et al. A whole-genome assembly of drosophila. *Science*, 287(5461):2196–2204, 2000.

[102] G. Myers. Efficient local alignment discovery amongst noisy long reads. In *International Workshop on Algorithms in Bioinformatics*, pages 52–67. Springer, 2014.

[103] N. Nagarajan and M. Pop. Sequence assembly demystified. *Nature Reviews Genetics*, 14(3):157–167, 2013.

[104] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[105] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[106] F. Nicolas and E. Rivals. Hardness of optimal spaced seed design. *Journal of Computer and System Sciences*, 74(5):831–849, 2008.

[107] S. K. Nielsen, T. L. Koch, F. Hauser, A. Garm, and C. J. Grimmelikhuijzen. De novo transcriptome assembly of the cubomedusa tripedalia cystophora, including the analysis of a set of genes involved in peptidergic neurotransmission. *BMC genomics*, 20(1):175, 2019.

[108] L. Noé and G. Kucherov. *YASS: Similarity search in DNA sequences*. PhD thesis, INRIA, 2003.

[109] L. Noé and G. Kucherov. Improved hit criteria for DNA local alignment. *BMC bioinformatics*, 5(1):149, 2004.

[110] L. Noé and G. Kucherov. YASS: enhancing the sensitivity of DNA similarity search. *Nucleic acids research*, 33(suppl_2):W540–W543, 2005.

[111] NVIDIA Corporation. Apex, 2019.

[112] J. Oh, A. L. Byrd, C. Deming, S. Conlan, B. Barnabas, R. Blakesley, G. Bouffard, S. Brooks, H. Coleman, M. Dekhtyar, et al. Biogeography and individuality shape function in the human skin metagenome. *Nature*, 514(7520):59, 2014.

[113] C. Olah. Understanding lstm networks. http://colah.github.io/posts/ 2015-08-Understanding-LSTMs/, 2018. Aug, 2015.

[114] Y. Ono, K. Asai, and M. Hamada. Pbsim: Pacbio reads simulator—toward accurate genome assembly. *Bioinformatics*, 29(1):119–121, 2013.

[115] Pacific Biosciences. E coli k12 mg1655 resequencing. https://github.com/PacificBiosciences/DevNet/wiki/E-coli-K12-MG1655-Resequencing, 2013. Jul 22, 2013.

[116] Pacific Biosciences. H. sapiens 10x sequence coverage with pacbio data, 2014.

[117] Pacific Biosciences. Smrt analysis. http://www.pacb.com/products-and-services/analytical-software/smrt-analysis/, 2014. Oct 15, 2014.

[118] PacificBiosciences. E. coli bacterial assembly primary analysis (instrument output) data, 2016. Accessed May 13, 2016.

[119] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[120] M. Pellegrini and T. O. Yeates. Searching for frameshift evolutionary relationships between protein sequence families. *Proteins: Structure, Function, and Bioinformatics*, 37(2):278–283, 1999.

[121] H. Peltola, H. Söderlund, and E. Ukkonen. Algorithms for the search of amino acid patterns in nucleic acid sequences. *Nucleic acids research*, 14(1):99–107, 1986.

[122] R. Poplin, P.-C. Chang, D. Alexander, S. Schwartz, T. Colthurst, A. Ku, D. Newburger, J. Dijamco, N. Nguyen, P. T. Afshar, et al. A universal snp and small-indel variant caller using deep neural networks. *Nature biotechnology*, 36(10):983, 2018.

[123] E. Prestat, M. M. David, J. Hultman, N. Taş, R. Lamendella, J. Dvornik, R. Mackelprang, D. D. Myrold, A. Jumpponen, S. G. Tringe, et al. Foam (functional ontology assignments for metagenomes): a hidden markov model (hmm) database with environmental focus. *Nucleic acids research*, 42(19):e145–e145, 2014.

[124] M. A. Quail, M. Smith, P. Coupland, T. D. Otto, S. R. Harris, T. R. Connor, A. Bertoni, H. P. Swerdlow, and Y. Gu. A tale of three next generation sequencing platforms: comparison of ion torrent, pacific biosciences and illumina miseq sequencers. *BMC genomics*, 13(1):341, 2012.

[125] S. Rajasekaran and M. Nicolae. An elegant algorithm for the construction of suffix arrays. *Journal of Discrete Algorithms*, 27:21–28, 2014.

[126] D. A. Rasko, D. R. Webster, J. W. Sahl, A. Bashir, N. Boisen, F. Scheutz, E. E. Paxinos, R. Sebra, C.-S. Chin, D. Iliopoulos, et al. Origins of the e. coli strain causing an outbreak of hemolytic–uremic syndrome in germany. *New England Journal of Medicine*, 365(8):709–717, 2011.

[127] J. Ren, K. Song, C. Deng, N. A. Ahlgren, J. A. Fuhrman, Y. Li, X. Xie, and F. Sun. Identifying viruses from metagenomic data by deep learning. *arXiv preprint arXiv:1806.07810*, 2018.

[128] A. Rhoads and K. F. Au. Pacbio sequencing and its applications. *Genomics, proteomics & bioinformatics*, 13(5):278–289, 2015.

[129] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[130] L. Salmela and E. Rivals. Lordec: accurate and efficient long read error correction. *Bioinformatics*, page btu538, 2014.

[131] T. Schiex, J. Gouzy, A. Moisan, and Y. de Oliveira. Framed: a flexible program for quality check and gene prediction in prokaryotic genomes and noisy matured eukaryotic sequences. *Nucleic acids research*, 31(13):3738–3741, 2003.

[132] S. Schwartz, W. J. Kent, A. Smit, Z. Zhang, R. Baertsch, R. C. Hardison, D. Haussler, and W. Miller. Human–mouse alignments with BLASTZ. *Genome research*, 13(1):103–107, 2003.

[133] S. Seo, M. Oh, Y. Park, and S. Kim. Deepfam: deep learning based alignment-free method for protein family modeling and prediction. *Bioinformatics*, 34(13):i254–i262, 2018.

[134] D. Sharon, H. Tilgner, F. Grubert, and M. Snyder. A single-molecule long-read survey of the human transcriptome. *Nature biotechnology*, 31(11):1009, 2013.

[135] J. T. Simpson and R. Durbin. Efficient construction of an assembly string graph using the FM-index. *Bioinformatics*, 26(12):i367–i373, 2010.

[136] I. Sović, K. Križanović, K. Skala, and M. Šikić. Evaluation of hybrid and non-hybrid methods for de novo assembly of nanopore reads. *Bioinformatics*, 32(17):2582–2589, 2016.

[137] I. Sović, M. Šikić, A. Wilm, S. N. Fenlon, S. Chen, and N. Nagarajan. Fast and sensitive mapping of nanopore sequencing reads with graphmap. *Nature communications*, 7, 2016.

[138] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[139] Y. Sun and J. Buhler. Designing multiple simultaneous seeds for DNA similarity search. *Journal of Computational Biology*, 12(6):847–861, 2005.

[140] H. Tilgner, F. Grubert, D. Sharon, and M. P. Snyder. Defining a personal, allele-specific, and single-molecule long-read transcriptome. *Proceedings of the National Academy of Sciences*, 111(27):9869–9874, 2014.

[141] B. J. Tindall, J. Sikorski, S. Lucas, E. Goltsman, A. Copeland, T. Glavina Del Rio, M. Nolan, H. Tice, J. F. Cheng, C. Han, S. Pitluck, K. Liolios, N. Ivanova, K. Mavromatis, G. Ovchinnikova, A. Pati, R. Faehnrich, L. Goodwin, A. Chen, K. Palaniappan, M. Land, L. Hauser, Y. J. Chang, C. D. Jeffries, M. Rohde, M. Goeker, T. Woyke, J. Bristow, J. A. Eisen, V. Markowitz, P. Hugenholtz, N. C. Kyrpides, H. P. Klenk, and A. Lapidus. Complete genome sequence of meiothermus ruber type strain (21). *Standards in genomic sciences*, 3(1):26–36, 2010.

[142] E. Trost, S. Götker, J. Schneider, S. Schneiker-Bekel, R. Szczepanowski, A. Tilker, P. Viehoever, W. Arnold, T. Bekel, J. Blom, et al. Complete genome sequence and lifestyle of black-pigmented corynebacterium aurimucosum atcc 700975 (formerly c. nigricans cn-1) isolated from a vaginal swab of a woman with spontaneous abortion. *BMC genomics*, 11(1):91, 2010.

[143] B. Trzaskowski, D. Latek, S. Yuan, U. Ghoshdastider, A. Debinski, and S. Filipek. Action

of molecular switches in gpcrs-theoretical and experimental studies. *Current medicinal chemistry*, 19(8):1090–1109, 2012.

[144] Y.-C. Tsai, S. Conlan, C. Deming, J. A. Segre, H. H. Kong, J. Korlach, J. Oh, N. C. S. Program, et al. Resolving the complexity of human skin metagenomes using single-molecule sequencing. *MBio*, 7(1):e01948–15, 2016.

[145] R. VanBuren, D. Bryant, P. P. Edger, H. Tang, D. Burgess, D. Challabathula, K. Spittle, R. Hall, J. Gu, E. Lyons, et al. Single-molecule sequencing of the desiccation-tolerant grass oropetium thomaeum. *Nature*, 527(7579):508, 2015.

[146] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[147] S. Vinga and J. Almeida. Alignment-free sequence comparison—a review. *Bioinformatics*, 19(4):513–523, 2003.

[148] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

[149] M. Wang, L. Tu, D. Yuan, D. Zhu, C. Shen, J. Li, F. Liu, L. Pei, P. Wang, G. Zhao, et al. Reference genome sequences of two cultivated allotetraploid cottons, gossypium hirsutum and gossypium barbadense. *Nature genetics*, 51(2):224, 2019.

[150] Q. Wang, J. F. Quensen, J. A. Fish, T. K. Lee, Y. Sun, J. M. Tiedje, and J. R. Cole. Ecological patterns of nifh genes in four terrestrial climatic zones explored with targeted metagenomics using framebot, a new informatics tool. *MBio*, 4(5):e00592–13, 2013.

[151] C.-L. Xiao, S. Zhu, M. He, D. Chen, Q. Zhang, Y. Chen, G. Yu, J. Liu, S.-Q. Xie, F. Luo, et al. N6-methyladenine dna modification in the human genome. *Molecular cell*, 71(2):306–318, 2018.

[152] E. M. Zdobnov and R. Apweiler. Interproscan–an integration platform for the signature-recognition methods in interpro. *Bioinformatics*, 17(9):847–848, 2001.

[153] H. Zeng, M. D. Edwards, G. Liu, and D. K. Gifford. Convolutional neural network architectures for predicting dna–protein binding. *Bioinformatics*, 32(12):i121–i127, 2016.

[154] Y. Zhang and Y. Sun. Hmm-frame: accurate protein domain classification for metagenomic sequences containing frameshift errors. *BMC bioinformatics*, 12(1):198, 2011.

[155] Z. Zhang, W. R. Pearson, and W. Miller. Aligning a dna sequence with a protein sequence. *Journal of Computational Biology*, 4(3):339–349, 1997.